THE HECKMAN BINDERY, INC.
North Manchester, Indiana

H or V | JUST | FONT | SLOT | TITLE

H CC 1W
GR
CULTY
ACTING
12 PAPER

H CC 1W 8 1988
7 NO 1468-1485

H CC 1W
CAS
B[illegible]V">
no.1468-1485
c.p.1

H CC 7W
U. of IL
LIBRARY
URBANA

BINDING COPY

PERIODICAL ☐ CUSTOM ☐ STANDARD ☐ ECONOMY ☐ THESIS
BOOK ☐ CUSTOM ☐ MUSIC ☐ ECONOMY AUTH 1ST ☐

ACCOUNT | LIBRARY | NEW | RUB OR SAMPLE | TITLE I D | FOIL | COLOR MATERIAL
66672 | WHT

ACCOUNT NAME
UNIV OF ILLINOIS

ACCOUNT INTERNAL I D | ISSN
D0191240.

I.D. #2 | NOTES | BINDING FREQUENCY | WHEEL | SYS I.D.
STX4 | 29276

COLLATING
36

ADDITIONAL INSTRUCTIONS
Dept=STX4 Lot=201 Item=135 PNM=[illegible]
1CR2ST3CR MARK BY # 84 91

SEP SHEETS | PTS BD PAPER | TAPE STUBS | CLOTH EXT | FILLER GUM | FILLER STOP
POCKETS: PAPER | BUCK | CLOTH
SPECIAL PREP | LEAF ATTACH

INSERT MAT | ACCOUNT LOT NO 201 | ORDER NO | PIECE NO 47
PRODUCT TYPE | ACCOUNT PIECE NO
HEIGHT | GROUP CARD | VOL THIS TITLE
COVER SIZE X

00124921

BEBR

FACULTY WORKING PAPER NO. 1485

An Integrated Framework for Applying Machine Learning in Intelligent Decision Support Systems

Michael J. Shaw

College of Commerce and Business Administration
Bureau of Economic and Business Research
University of Illinois, Urbana-Champaign

BEBR

FACULTY WORKING PAPER NO. 1485

College of Commerce and Business Administration

University of Illinois at Urbana-Champaign

September 1988

An Integrated Framework for Applying Machine Learning in Intelligent Decision Support Systems

Michael J. Shaw, Assistant Professor
Department of Business Administration

Abstract

Recently, research efforts have emerged to integrate the three functions of decision support systems (DSS)--i.e., database management, model management, and problem solving--in the knowledge-based expert system environment. This paper is aimed at adding the learning capability as a new dimension to the operational design of the knowledge-based DSS, taking the view that the ability to learn is essential for every intelligent system.

We will show the machine learning techniques for acquiring decision rules, refining decision knowledge, and enhancing model-based query processing in the DSS. The machine learning techniques used include: (1) inductive learning methods for learning classification rules and decision heuristics, for which the performance characteristic of different inductive learning methods are compared; (2) the learning-from-experimentation method for learning model-management knowledge; and (3) the learning-by-explanation method for learning problem-solving schemata.

1. Introduction

There are three key functions of decision support systems (DSSs): (1) supporting comparatively unstructured decision activities by furnishing the system's users with powerful yet simple to use problem solving tools, (2) managing the data necessary for the decision tasks, and (3) maintaining models and applying these models to support complex decisions (Bonczek [1981]). Recently, research efforts have been emerging that tailor the knowledge-based expert system methodology for designing DSSs (Henderson [1987], Stohr [1985]). Besides the usual deductive inference mechanism, this paper will add a new dimension to the operational design of knowledge-based DSSs: the ability of the system to learn.

Recognized as the essential feature of any intelligent system, learning processes include the acquisition of new declarative knowledge, the development of problem-solving skills through instruction or practice, the organization of new knowledge into general, effective representations, and the discovery of new facts and theories through observation and experimentation. Machine learning is concerned with the computer modeling of the learning processes. In this paper we will discuss several aspects where machine learning techniques can improve the operations of DSSs, namely: (1) learning decision rules for the DSS's knowledge base; (2) learning to refine knowledge by observing prior problem-solving processes; and (3) learning to perform multiple-step DSS tasks, where data retrievals and model manipulations are involved.

The DSSs discussed in the literature mostly employ only static decision knowledge provided by domain experts. In some DSS applications, however, it may not be possible to have perfect, predetermined decision knowledge. The decision-support task involved in assisting business loan evaluation is a case in point. After interviewing several bankers in Chicago in an effort to develop a DSS for business loan evaluation, we found that the loan-evaluations process is still conducted in a primitive fashion. The loan officers often are not aware of exactly which decision rules they are applying. Moreover, it is difficult to get the concensus among officers about the standard rules to use. As a result, evaluation decisions are frequently based on vaguely defined classifications of financial data and sometimes can be arbitrary. By contrast, a decision-support system with learning abilities can derive rules for loan-granting decisions directly from observing prior decision examples.

Moreover, the solution processes for tasks such as loan evaluation typically need the support from a large amount of data (e.g., financial data) and appropriate models (e.g., program modules for forecasting and optimization). Accordingly, the DSS usually has to resort to a sequence of information-processing activities, forming a plan for using the embedded knowledge base, data base, and model base for getting the solution. To this end, we shall show methods for learning the formation of such solution processes when multiple steps of inferences, data retrievals, and model applications are involved. Such learning processes will be characterized as the acquisition and the refinement of "problem-solving schemata" in this paper.

The remainder of the paper is organized as follows. Section 2 addresses the DSS as an integrated problem-solving environment and presents a framework in which machine learning can be incorporated; Section 3 describes the process of inductive learning and shows the examples of using inductive learning to derive decision rules; Section 4 applies a machine learning approach to model management, focusing on the learning of model-manipulation schemata and the refinement of modeling knowledge; finally, Section 5 describes the explanation-based learning method for learning schemata.

2. A New DSS Framework Incorporating Machine Learning

Machine learning methods can be categorized into the following areas based on their behavioral characteristics: rote learning (Samuel [1968]), learning from instruction (Davis [1979]), learning by induction (Buchanan and Mitchell [1978], Dietterich and Michalski [1983]), learning by analogy (Winston [1979], Carbonell [1983]), learning by competition (Holland [1986]), and learning from observation and discovery (DeJong [1986], Langley [1981], Lenat [1983]).

A basic machine learning model is summarized in Figure 2.1, where the learning system consists of four elements: Environment, Learning Element, Knowledge Base, and Performance Element. The Learning Element takes its input from the Environment, in the form of observations, or from the Performance Element, in the form of performance results; the learning process will result in either new knowledge for the knowledge base or modifications on the existing knowledge. We shall adapt this basic model to the intelligent DSS setting, where the

input from the Environment is collected from a firm's database, and the Performance Element corresponds to the rule-based problem solver of the DSS.

Insert Figure 2.1 Here

The Knowledge Base in the DSS setting contains: (1) procedural knowledge, (2) decision heuristics, and (3) model-manipulation knowledge. Procedural knowledge is the knowledge about the essential steps, mostly related to information collection, for making a given decision; for example, to evaluate a company's credit-worthiness, the necessary supporting information includes the competence of the management, the outside credit rating of the firm, and credit analysis on the firm's financial data. The decision heuristics are rules of thumb used by domain experts. Because of the inherently judgmental nature, this type of rules needs considerably more effort to obtain and refine. The rules generated by inductive learning belong to this category. The third type of rules is used to represent the model knowledge available for decision support; these rules indicate the application requirements of each model and the relations between models. Some examples of rules of this type are shown in Appendix 2.

Most existing DSSs use knowledge engineering for acquiring problem-solving knowledge; they take the domain knowledge from experienced decision makers in the field and transform the knowledge into the representation used in the knowledge base of the DSS (Elam and Henderson [1980], Stohr [1986]). This is shown as process (a) in

Figure 2.2. There are two aspects of rule learning for the knowledge base: (1) Learning from an example set, in which decision rules are derived from a given set of positive and negative examples (shown as process (b) in Figure 2.2); and (2) Rule modification, in which the rules in the knowledge-base are modified to improve the performance of the DSS (shown in process (c) in Figure 2.2). Learning from examples can be achieved by inductive inference (Rendell [1986], Michalski [1983]). Rule refinement, on the other hand, can be achieved by comparing the resulting solution path (i.e., the performance trace) with the correct path (i.e., the ideal trace). Bundy [1985] reviewed several methods for rule refinement and compared their performances.

Insert Figure 2.2 Here

Our approach incorporates four interactive functional components--the Instance Selector, Problem-Solver, Critic, and Learning Module--to integrate the learning functions. The Instance Selector either accepts the training instances supplied externally or generates new training examples by itself in response to previous learning processes. The Problem-Solver produces solutions to the new problems supplied by the Instance Selector by applying existing knowledge stored in the knowledge base. The resulting solution path for each new problem is then evaluated by the Critic, which compares the solution just produced with the desired solution. Based on the observations made by the Critic, the Learning Module either refines existing rules or hypothesizes new rules. This learning augmented DSS configuration for knowledge refinement is shown in Figure 2.3.

Insert Figure 2.3 Here

---

3. Inductive Learning for Acquiring Decision Rules

3.1 Concept Learning

Inductive learning can be defined as the process of inferring the description of a class from the description of individual objects of the class. Training examples are given in the form of cases and described by a vector of attribute values. Each class can be viewed as a concept which is described by a rule determined by inductive learning. If an input data case satisfies the conditions of this rule, then it represents the given concept. A concept is a symbolic description expressed in some description language that is true when applied to a data case describing the concept correctly and false otherwise. For example, a recognition rule for the concept "class IA firm" might be:

> "A firm whose asset exceeds $1,000,000.00, total debt is less than $250,000.00, and whose annual growth rate is more than 10%."

Using first-order predicate calculus (FOPC) as the knowledge representation, the same concept can be represented by a conjunction of attribute descriptions:

> customer (t) & (asset (t) > $1,000,000) & (total-debt (t) < $250,000) & (AGR(t) > 0.10) → (class (t) = 'IA')

An alternative way to represent such a concept is exemplified by the variable-valued logic (VL) proposed by Michalski [1983]. The

aforementioned concept recognition rule can be represented by the VL formalism as follows:

[assets > $1,000,000] & [total-debt < $250,000] &

[AGR > 0.10] → [class : 'IA'].

An instance that satisfies the concept definition is called a positive example of that concept, whereas an instance that does not satisfy the concept definition is called a negative example of that concept. A generalization of an example is a concept definition which describes a set containing that example. For a set of training examples, the generalization process identifies the common features of these examples and formulates a concept definition describing these features. Thus, inductive learning can be viewed as a process of repetitively generalizing the descriptions observed from examples until the inductive concept definition is found. This resulting concept must be consistent with all the examples.

The input to an inductive learning algorithm consists of three parts: (1) a set of positive and negative examples, (2) generalization rules and other transformation rules, and (3) the criteria for a successful inference. Each training example has two components: first, a data case consisting of a set of attributes, each with an assigned value; the second component, on the other hand, is a classification decision made by a domain expert according to the given data case. The output generated by this inductive learning algorithm is a set of decision rules consisting of inductive concept definition for each of the classes. Learning programs falling into this category include AQ-Star (Michalski [1983]), PLS (Rendell [1986]), and ID3

(Quinlan [1979]). These programs are sometimes referred to as "similarity-based" methods, as opposed to explanation-based methods (Mitchell et al. [1986]). The discussions in this section focus on the former. The explanation-based learning technique for decision support will be presented in Section 5.

3.2 The AQ-Star Method

Developed by Michalski and his research group, the AQ-Star method is based on similarity-based learning and uses the variable-valued logic as the representation language. The input to the AQ-Star program consists of three parts: (1) a set of positive and negative examples, (2) generalization rules and other transformation rules, and (3) the criteria for a successful inference. Each training example has two components: first, a data case consisting of a set of attributes, each with an assigned value; the second component, on the other hand, is a classification decision made by an experienced loan officer according to the given data case. The output generated is a set of decision rules consisting of inductive concept definition for each of the classes.

Let e be an example of a concept to be learned and E be a set of counterexamples of the same concept. A set of descriptions is said to cover the example if they are satisfied by example e. A star of the example e against the example set E, denoted $G(e|E)$, is defined as the set of descriptions that cover example e and that do not cover any of the negative examples in E. Figure 3.1 illustrates an example of such a star for a positive example e against the set of negative examples

E. The star in this case consists of three descriptions A, B, and C. The concept of a star is useful because it reduces the problem of finding a complete description of a concept to subproblems of finding descriptions of single positive examples; the set of negative examples are used as constraints confining the descriptions, so that none of the negative examples would be covered by any elements of the star.

Insert Figure 3.1 Here

Let S_p and S_N represent the sets of positive and negative examples, respectively. The inductive learning algorithm based on the star methodology can be described as follows:

The AQ-Star

Input:

(i) the set of positive examples;
(ii) the set of negative examples;
(iii) the set of generalization rules and other transformational heuristics;
(iv) the preference criteria for selecting the promising descriptions among alternatives.

Output:

Begin

1. Randomly select an example e from S_p.
2. Generate a star, $G(e|S_N)$, of the example e against the set of negative examples S_N. If applicable, generalization rules or other heuristics can be applied in this star-generation process.
3. In the obtained star, find a description D with the highest preference according to the specified preference criterion.

4. If D covers set S_p completely, then go to step 6.

5. Otherwise, reduce the set S_p to contain only examples not covered by D, and repeat the whole process from step 1.

6. The disjunction of all generated descriptions D should cover all the examples in S_p and none in S_N. Exit.

End.

Like most AI programs, the Inductive Learning Algorithm also suffers the problem of combinatorial explosion. The major culprit is step 2, the process of star generation. In most applications, a star of an event may contain a very large number of descriptions. A "beam-search" method can be used to alleviate this problem. The beam-search method limits the number of descriptions in a star to a predetermined number m, referred to as the beam size.

We can use the AQ-Star program as an example to illustrate the process of rule learning. Suppose that the data shown in Table 3.1 are part of a set of credit rating data serving as training examples for learning the concept for risk classification. The data set contains historical and per forma financial information belonging to nine companies, each with an assigned risk class. Let's suppose that companies A, B, and C are known to be in Class I; companies D, E, and F are in Class IA; and companies G, H, and I are in Class II.

Insert Table 3.1 Here

Two types of generalization rules are used in the learning process: domain specific rules and domain independent rules. An example of the domain specific rules used is

[Account-type = commission] V [Account-type = fees]-->

[Account-type = other-businesses].

This rule indicates that [Account-type = other business] is a generalization for either [Account-type = commission] or [Account-type = fees]. In addition, there is a set of domain independent rules for generalization. There are a variety of generalization rules of this type, such as the closing-interval rule and the dropping-condition rule (Michalski [1983], p. 106).

The induction criteria used for this example are (1) to cover all of the positive examples, while not covering any of the negative examples, and (2) to include the least number of attributes in the concept definitions.

The AQ-Star inductive learning algorithm is then applied to the set of example data, resulting in the following three inductive rules corresponding to the concept definition for the three classes:

1. [avg-inventory $\geq$ \$7,000] & [net-worth $\geq$ \$47,000] --> [class = I].
2. [\$37,000 $\leq$ net-worth $\leq$ \$48,000] & [inventory > \$8,000] --> [class = IA].
3. [Management-rating = High, Average] & [total-debt $\geq$ \$26,000] --> [class = II].

The resulting three decision rules generated can then be used as decision rules for risk classification. These classification rules cover all the positive examples but none of the negative examples (These two induction criteria are referred to as (1) the completeness and (2) the consistency conditions in Michalki [1983].).

3.3 The ID3 Method: Induction of Decision Trees

ID3 is an inductive learning program following Hunt's earlier work (Hunt et al. [1966]). ID3 takes data cases of a known class described in terms of a fixed set of attributes, and produces a decision tree over these attributes that correctly classifies the given cases.

ID3 generates decision trees based on an information theoretic approach. Let the set S contain p cases of class P (i.e., set S_p) and n of class N (i.e., set S_N). Then, assuming that a given data case may belong to class P with probability p/(p+n) and to class N with probability n/(p+n), the amount of information needed to produce the message 'P' or 'N' is measured by

$$I(p,n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}.$$

The decision tree is generated by, starting with a root node, progressively selecting attributes to branch the tree. At each node, the selection of an attribute A_i to branch will partition the tree into M subtrees, where M is the number of values of A_i when A_i is a nominal variable; otherwise a binary split of A_i is carried out in which case M = 2. Let the kth subtree contain p_k cases of class P and n_k cases of class N. The expected information required for the tree with A_i as the root is

$$E(A_i) = \sum_{i=1}^{m} \frac{p_i+n_i}{p+n} I(p_i,n_i).$$

The information gained by branching on A_i is therefore

$$\text{Info-gain } (A_i) = I(p_i,n_i) - E(A_i).$$

At each iteration of generating the decision tree, ID3 examines all candidate attributes and chooses the attribute that can maximize the amount of information gained. The basic algorithm for the induction of decision trees is as follows:

Algrotihm INDUCE-TREE

Begin

1. Create the root node.
2. If all examples at the current node are of the same class, stop.
3. For each attribute A_i, compute the value of gain (A_i).
4. Choose the attribute with the highest gain (A_i) to branch the current node.
5. For each branch node, go to step 2.

End

For handling large sets of training examples, Quinlan extended the INDUCE-TREE procedure and incorporated a 'window' scheme that deals with a subset of training examples at a time.

Algorithm ID3

Begin

1. Select at random a subset of the training examples (called the window).
2. Repeat
3. Call INDUCE-TREE to form a tree to explain the current window.
4. Find the exceptions to this tree in the remaining examples.

5. Form a new window from the current window and the exceptions to the tree generated from it.

6. Until there are no exceptions to the tree.

End

Using the data shown in Table 3.1 as training examples, ID3 generates a induction tree as depicted in Figure 3.2.

Insert Figure 3.2 Here

3.4 Probabilistic Learning System (PLS)

The inductive process followed by the PLS algorithm starts with the entire space of possible events (the "feature space"). The space is then further split into two "regions," those of which have a greater likelihood to being in a specific class (positive events) and those which have a greater likelihood to being in the other classes (negative events). The process of splitting continues, each split using only one attribute that is chosen according to an information-theoretic approach, until a stopping criterion is satisfied. In each iteration, the region R in the feature space can be defined by R = (r,u,e), where r is a hyper-rectangular region in the feature space; u is a utility function, and e is the error rate allowed in a region. The information-theoretic approach is based on the region's utility which is the fraction of positive events in that region. Since the purpose is to maximize dissimilarity within a feature space, the split is made based upon maximizing the difference in the utilities of the two regions (known as the distance function). Each region, also

called a hyper-rectangle, is also associated with its error measure e. In proportion to the goodness of the utility, e has a lower value. The distance function (d) is defined as follows:

$$d = |\log u_1 - \log u_2| - t^* \log(e_1 * e_2)$$

where

u_1, u_2 - probabilities for a tentative region dichotomy,

e_1, e_2 - respective error factors,

t - a constant representing the degree of confidence.

Larger values of d correspond to higher dissimilarity.

The sub-hypotheses of a class represents a conjunction for the hyper-rectangular equation for a region; a disjunction of the sub-hypotheses is defined as the hypotheses for the class of events.

PLS is data driven, and is not sensitive to noise since it is stochastic. The PLS algorithm has been used in domains such as medical diagnosis, and also in the application of heuristic search, among others.

Let S be the set of positive and negative training events and R as the hyper-plane that contains all events in E, the PLS algorithm can be summarized as follows:

Algorithm PLS

While any trial hyper-plane remain untested, do

Begin

1. Choose a hyper-plane not previously selected to become a tentative boundary for two subrectangles of R, r1 and r2.

2. Using the events from S, determine the utilities u1 and u2 of r1 and r2, and their error factors e1 and e2 (where u is the conditional probability that an instance of E falling within region i is a positive event).

3. If this tentative dichotomy produces a dissimilarity d larger than any previous value for d, the corresponding region, and the associated values of u1 and u2:

   If d > 0 (i.e., if the maximum dissimilarity is significant);

   then create two permanent regions R1 = (r1, u1, e1) and R2 = (r2, u2, e2) having the (previously recorded) common boundary that gives the most dissimilar probabilities;

   else place R in the defined region set R to be output, and quit.

End.

Note that each region in the set R represents a sub-hypothesis, with a disjunction of the regions in R representing the target hypothesis which describes the events in E.

For the sake of comparison, we apply PLS to the same set of training data shown in Table 3.1, and obtain the concept descriptions depicted in Figure 3.3.

Insert Figure 3.3 Here

3.5 An Empirical Study

To test the performance of the inductive inference method for rule learning in the DSS domain, we have conducted an empirical study using real-world data for risk analysis. This study uses financial data for predicting bankruptcy. The task for the inductive inference engine is to perform concept learning about the characteristics of bankrupt firms. Three inductive learning programs, AQ-Star, ID3, and PLS are used for generating rules.

The data are obtained from the bankruptcy study reported in Gentry, et al., [1985]. The Standard and Poor's Compustat 1981 Industrial Annual Research File of companies, and the Compustat Industrial Files were used to determine companies that failed during the period 1970-81. Balance sheet and income statement information for the failed companies was used to determine the funds flow components. There are a total of eight funds flow components used as the attributes for each data case. There were a total of 29 companies of which the complete financial statement information for the year before the failure date was available. These companies are used as positive examples. Furthermore, each of the 29 failed companies was matched with a nonfailed company in the same industry, based on the asset size and sales for the fiscal year before bankruptcy. The same set of financial data are provided for each of these nonfailed companies, which serve as negative examples of the concept. The objective of the analysis is to determine whether inductive learning can effectively discriminate between failed and nonfailed companies by the financial data one year ahead of failure. The rule learning program is written in PASCAL on VAX 11/780.

The set of training examples are the funds flow components of the failed and nonfailed firms. To test the predictive accuracy of the rules generated by the inductive learning algorithm, we use the holdout sample technique and use half of the sample for rule learning; the rules are then tested on the remainder of the sample. The selection of training examples out of the set of data is based on a degree of representativeness of each data case.

Insert Table 3.2 Here

The result of using the learned rules to test against the holdout sample is shown in Table 3.2 (Piramuthu and Shaw [1988]), which shows that the learned rules are quite effective in predicting and classifying; the results compare favorably with 83.3 percent accuracy resulting from the logit model used in Gentry, et al., [1985]. (Note that, as in Rendell's (Rendell et al. [1987]) study, the examples tested are partitioned into positive and negative examples.) Although this empirical study is preliminary in the sense that only one set of data is used, it nevertheless shows that the rules generated by inductive learning provide a valid decision aid for classification problems, such as determining whether a given firm has the characteristics of bankrupt firms.

There are a number of advantages associated with applying inductive learning techniques to this type of data analysis tasks. First of all, the representation is clearer than the linear model traditionally used and, as a result, the amount of information presented is richer. For example, the induction tree generated by the ID3 system is shown in Figure 3.4. Such a decision tree not only provides a means for classifying data cases, it also indicates the relative importance of the attributes--and the attributes irrelevant to the classification would not be selected by the algorithm.

Insert Figure 3.4 Here

3.6 Comments

The performance results obtained from the empirical study are quite consistent with those reported in other empirical studies. In particular, O'Rorke [1982] and Michalski [1986] showed that the classifications produced by ID3 are more accurate than the ones produced by AQ. Rendell [1987] further showed that PLS gives more accurate classification than ID3 and AQ. Moreover, PLS also performs better when noisy data exist. Figure 3.5, taken from Rendell et al., [1987], shows the classification accuracy with class error and attribute error, respectively, for the aforementioned three systems. In view of these empirical results, a logical question to ask is: "What are the major design factors of an inductive learning program that would have an impact on the learning performance?" Furthermore, to put the question in a precise context, it would also be necessary to define the criteria for evaluating the learning performance. We shall briefly address these two issues to conclude our discussion on similarity-based learning methods.

Insert Figure 3.5 Here

Dietterich & Michalski [1983] and Rendell [1987] pointed out a number of evaluation criteria for inductive learning programs, such as classification accuracy, adequacy of the representation language, conciseness of concept descriptions, auxiliary domain information, computational efficiency, flexibility, and ease of use. In a more recent empirical study, Rendell [1987] summarized several design characteristics of inductive learning programs, three of which are particularly useful:

(1) Inductive Operators

AQ starts with a single example case which is treated as a seed hypothesis for the generalization process. By contrast, both ID3 and PLS perform specialization. ID3 does so by repetitively selecting attributes for branching the decision tree; PLS by splitting a region into subregions.

(2) Concept Representation

AQ-Star uses a logic description language called VL1 to represent concepts. ID3 employs the decision-tree representation. In PLS, input cases are represented as points in a k-dimensional event space; concepts are represented as orthogonal hyper-rectangles, called regions, in the event space.

(3) The Hypothesis Form

Both ID3 and PLS use probability information in the process of specialization. In ID3, this probability information is the measurement of information gains from branching an attribute. In PLS, an error term is used to indicate the system's confidence level regarding the clustering of the event space. Thus, both systems have graded hypothesis forms. AQ-Star, on the other hand, uses a specialization mechanism that treats class memberships as binary relationships. Its hypothesis form is not graded.

These three design characteristics may help shed light on the relative performance of the inductive learning methods. For example, we have seen that the AQ-Star method is more susceptible to noisy data than the other two methods. This can be explained by the fact that

the class-membership in AQ-Star is treated as strictly binary, whereas ID3 and PLS treat class-membership probabilistically. As a result, AQ-Star tends to misclassify a case with noisy data; on the other hand, ID3 and PLS can still maintain the "more likely" classification for the data cases tainted by noisy data.

Computation efficiency is another consideration. Because the way AQ-Star generates bounded stars in the learning process, the complexity grows exponentially. By contrast, both Quilan [1983] and Rendell et al., [1987] reported linear computation time for ID3 and PLS, respectively. O'Rorke [1982] also confirmed that ID3 is faster than AQ-Star. In terms of computation complexity, let ℓ be the number of attributes, j be the number of data cases, Quilan [1986] showed that the complexity of ID3 is $O(\ell \cdot j)$. Rendell et al., [1987] showed that the complexity of PLS is $O(\ell \cdot j \cdot k)$ where k is the final number of regions. By treating the learning process as a specialization process and guiding the process by the information-theoretic heuristic, one can conclude that ID3 and PLS are more computationally efficient than AQ-Star. Lee & Ray [1986] followed up on this view and developed an interesting scheme, similar to ID3, which incorporates a branch-and-bound procedure in the process of specialization.

4. Applying Machine Learning to Model Management

Due to the characteristics of DSS applications, the solution process usually involves a great deal of trial-and-error, and data is transformed in various ways through a diverse collection of program modules--i.e., models. It is therefore necessary to have not only a

comprehensive collection of such models, but also suitable mechanisms to use these models effectively in problem solving. In other words, the design of the model management subsystem has a major impact on the interaction between a user and the DSS (Sprague and Carson [1982]). Since model applications are mostly judgemental, it is desirable for the DSS to learn the characteristics of various models and the applicability of each model for different types of problems, so that the way the models are used can adapt to the user's preference or the problem's characteristic. The model-management methods described in the literature to date (Applegate et al., [1986], Bonczek et al., [1986], Dolk and Konsynski [1984], Dutta [1984], Elam and Konsynski [1987], Fedorowicz and Williams [1986], Konsynski and Sprague [1986], Liang [1987], Menon and Shaw [1988]) have not addressed this learning aspect.

The major applications of machine learning to model management addressed in this paper are in three aspects: (1) the acquisition of model manipulation knowledge, (2) the refinement of model manipulation knowledge, and (3) the creation of model selection heuristics.

(1) The Acquisition of Model Manipulation Knowledge

Model manipulation involves selecting, retrieving, and activating models to solve problems; the individual models often need to be combined with one another into a sequence of problem-solving steps in order to reach the solution. A learning augmented model management system should be able to learn the solution procedure and store the solution procedure as a generalized piece of knowledge referred to as

a schema. Model manipulation knowledge can be learned by acquiring and refining such schemata.

We use the term model manipulation schemata to represent the knowledge learned from multi-step problem-solving processes. Every schema contains a condition part which describes a class of applicable problems and a solution part which displays the shared solution plan for every problem in this class. The solution plan can be represented by an AND/OR tree structure, which we call the solution tree. An OR subtree in the solution tree denotes all possible alternative solution paths, and an AND subtree indicates the input requirements for a model or a set of subproblems for a decomposable problem. The subproblems can be simple data retrievals or model executions. It should be noted that several models may generate similar solutions to a problem which altogether constitute an OR tree for this problem. Each subtree converging to an OR node is an alternative solution plan to this problem. Nodes in the bottom of the solution tree are either solvable terminal nodes which are subproblems solved by either data-retrieval or user input, or they may represent unsolvable terminal nodes which are subproblems that cannot be solved by simple data-retrievals, user input or models. The solution tree with at least one solution path whose terminal nodes are all solvable is complete since this solution tree can provide a solution plan to the problem. Otherwise, it is incomplete since it cannot provide any solution plan to the problem. The solution of a stored model manipulation schema is applicable only if a new problem matches with the condition part of this schema. The application of a model manipulation schema in the form of an AND/OR tree is shown in Figure 4.1. We use the schemata as problem solving

concepts. That is, useful schemata will be those that organize operators to achieve an important goal, or a set of goals, in a general way.

As the model management system usually deals with executing two or more models in an appropriate sequence, the process of model manipulation involves a multiple-step process, in which each step involves either a database retrieval or a model application. As opposed to searching for the individual steps, a learned model-manipulation schema integrates the entire multiple-step process into a single module; such a schema can be applied either as a single step or just a portion of it, depending on the problem to be solved. The schemata help reduce the amount of search required in solving similar problems, as shown in Figure 4.2.

The learning of model manipulation schemata can be characterized as "learning of multiple-step tasks," which is also used in Fikes [1972], Korf [1982], and DeJong [1986]. Moreover, the concept of model manipulation schemata is similar to the "macro-operators" used in Fikes [1972] and Korf [1982] for representing the sequence of actions learned. The learning procedure using the learning components in Figure 2.2 for acquiring model manipulation knowledge is depicted in Figure 4.3. We will show, in Section 5, an explanation based learning (EBL) technique for learning schemata. EBL is characterized by its use of a structured set of domain knowledge and a generalization process based on a single observation (Mitchell et al. [1986]).

Insert Figures 4.1, 4.2 and 4.3 Here

In addition to automatically acquiring model manipulation schemata, machine learning techniques also enables the model management system to refine these schemata after an iterative experimentation process. We shall elaborate on this experimentation process next.

(2) Refinement of Model Manipulation Knowledge

The model manipulation schema is derived from generalizing a specific problem instance and its solution plan. However, such a generalization may cover more than it is supposed to and, therefore, further refinement is often desirable. To increase the accuracy of the initially learned schema, the model management system needs to modify the schema through a training process which contains a collection of self-created or teacher-provided training examples. Since the system would choose and manipulate the training instances (by the Instance Selector) in order to verify the hypotheses about the concept, this process is sometimes referred to as learning by experimentation (Mitchell, Utgoff, and Banerji [1983]). The series of experiments with training instances would help the learning process converge to the correct concept description.

As defined in the preceding section, a schema consists of two parts: a condition part describing a class of problems to which this schema is applicable, and a solution part which displays the shared solution plan for every problem in this class. An experiment with a training instance provides a positive or a negative example for the current schema. A positive example has a complete instantiated solution plan based on the schema. A negative example, on the other hand,

is a problem instance which does not belong to the class under consideration. After the model manipulation schema is acquired by generalizing the derived solution plans for the given problem, the refinement of the schema on the current over-generalized form is achieved by an iterative process generalizing or constraining the training examples. This refinement process can be summarized by the following two operations: If the current problem expression of the schema does not cover the encountered positive example, then it needs to be generalized. If the current problem description of the schema covers the encountered negative example, then it needs to be constrained. This refinement process can be facilitated by organizing the possible problem descriptions in a "version space" (Mitchell [1982]).

Essentially, we are treating the refinement process as a search process for concept learning--in this case, the concept to be learned is the correct problem description for the schema. The search is conducted in the space of all possible versions of the descriptions, referred as the version space. The version space basically provides a generality/specificity structure for guiding the refinement process. Given a version space and a description in the version space, the Learning Module should be able to find the more generalized version of the description, the more specific version of the description, or descriptions belonging to the same level of generalization.

The approach described in Mitchell [1982] takes advantages of the general-to-specific ordering of descriptions in the version space. Mitchell argues that a version space can be represented by two sets of

descriptions, S and G, where S is the set of the most specific descriptions consistent with the observed instances, and G is the most general descriptions consistent with the observed instances.

To refine a model manipulation schema, we first create a version space for the problem descriptions to which this schema is applicable. This version space is represented by the G and S sets. Initially, G and S are defined by the first training example t_o: G is the maximal generalization of t_o and S is defined to be t_o. The set of training examples are then input sequentially to shrink the version space. For each input example, if the training example is a positive instance, then (1) generalize S, as little as possible, in order to cover this positive example and (2) remove from G all concepts that do not cover the example. On the other hand, if the training example is negative, then (1) remove from S the parts which cover the example and (2) make G more specific, as little as possible, so that its elements would not cover the example. Thus, in each step G is constrained to avoid covering the negative examples and S is made more generalized to cover the positive examples encountered. When G and S finally converge, the proper problem description for the schema is found. The learning procedure for the refinement of model manipulation knowledge is depicted in Figure 4.4.

Insert Figure 4.4 Here

As shown in Figure 4.4, it is sometimes necessary for the Instance Selector to <u>generate</u> training examples to expedite the schema refinement process. The main idea is to make some slight changes on a prior

training example and see if the changes would result in a different classification for the new example. By considering the new training example in the Learning Module, the S and G sets would move closer to each other. This converging process between S and G can be further facilitated if the new training examples selected represent concepts closely related to some prior training examples. Mitchell et al. [1983] has a more detailed account on how the new training examples can be generated by the Instance Selector to facilitate the learning process.

In many learning situations, it is possible to have "near miss examples"--i.e., the instances which are very close to being positive (p. 392, Winston [1984]). In refining model manipulation schemata, for example, the near miss examples can be defined as the problem descriptions which, although not directly solvable by the schema, need only minor modifications for the schema to be applicable (e.g., one unsolvable node in the solution tree when the schema is applied). The Learning Module can decide to modify the schema by finding the solution for the unsolvable node, so that it becomes applicable to this near miss example. The G and S sets are updated by treating the near miss example as a positive instance for the modified schema. The refinement process would continue until G and S converge. An example of schema refinement using positive, negative, and near-miss examples is described in Appendix 3.

(3) The Creation of Model Selection Heuristics

When there are more than one way to solve a given problem (e.g., models such as regression, moving average, exponential smoothing, and

delphi models can all solve a forecasting problem), the model management system usually either let the user select the best model, or it can choose among these alternative models based on a heuristic function. This heuristic function is chosen based on past performances or human experts' experiences and is usually in the form of a polynomial of several important factors: $E = \sum_i w_i * f_i$, where w_i is the weight given to f_i. For example, the f_is' used for scoring the forecasting models could be the forecasting accuracy, the operating cost, the operating time, and the difficulty of collecting data for each model, where each f_i is characterized in a numeric scale.

The coefficients of the heuristic function may be affected by the preference of the users and by the characteristics of the problems. A marketing manager may think that the accuracy of a forecasting model should dominate other criteria, but an MIS manager may give higher preference to the computational efficiency. As a result, different users may assign different heuristic functions to the models. Hence, the model management system should be able to adjust the coefficients of the heuristic function according to the "preference patterns" manifested by the users. To that end, an inductive learning method is needed to derive the heuristic function for each user, based on observations of that user's selection behavior. Since the objective is to make model selection <u>adaptive</u> to the user's preference, we adopted the inductive learning method articulated in Rendell's PLS described in Section 3.4. The heuristic function in this learning method corresponds to the utility function defined on a feature space. Recall that the feature space consists of a set of rectangular regions, each

of which contains instances of a single concept (i.e., class) and is defined as R = (r, u, e), where r is a rectangular region in the feature space; u is utility function value, as estimated by the probability given by the ratio of the positive instances to the total observed instances in this region; and e is the error rate allowed in this region. The utility indicates the probability that an instance in the region is a positive instance; e is used to represent the system's confidence in its judgement of the instances contained inside a region.

This PLS framework can be applied to generate the heuristics for model selection. For example, suppose that the models are ranked on two performance criteria: (1) quality of solution and (2) computational complexity, which are treated as the two dimensions of the feature space. Each region in the feature space then contains those model instances on the same utility level, described by the combination of solution quality and time complexity. For a given model, the corresponding utility--which represents the value for the model selection heuristics--can be determined by mapping it into the feature space. This approach progressively refines the utility assigned to each region by splitting a region into smaller regions. In addition, unlike some of the other learning system (e.g., Michalski [1983]), this learning method can effectively handle noisy data in the training set.

For example, in Figure 4.5a, the three problems--"the sales next year," "the inventory three years later," and "the interest expense next year"--all face the decision of choosing the best forecasting

model. For the sake of simplicity, we use the quality of solution and time complexity as two criteria for evaluating alternative models. In the set of training examples, the chosen model for each problem is treated as a positive instance, and the rest are treated as negative instances.

Insert Figure 4.5 Here

Using each model's solution quality and time complexity, the Learning Module can localize the models in the feature space (Figure 4.5b). To determine the heuristic value, the Learning Module further divides the feature space into several classes using the following procedure. Initially, it arbitrarily splits the feature space into two regions, and calculates the success probability, u, in each of these regions. In figure 11.b, an arbitrary splitting generates the regions, r_1 and r_2, which initially have the utilities (probabilities) $u_1 = 2/7$, and $u_2 = 1/5$, respectively; they are estimated by the ratio of positive instances to total instances in each region. Each region is then refined by a further splitting, where the best splitting is the one resulting in the largest dissimilarity d among all possible partitions of the region. Rendell defined the dissimilarity measure, d, for each splitting as $(|\log u_1 - \log u_2| - \log(e_1/e_2))$, where u_1, u_2, and e_1, e_2, are the utilities and error rates for the two regions after the splitting. This splitting process is repeated until $d \leq 0$ for every region, shown as Figure 4.5c in the example. Every region can then define a utility class, in which the models are of the same preference level. This inductive learning process can be applied to

the training examples collected from the individual users; the utility classification derived from a set of training examples can be used as the heuristic for model selection.

5. Explanation Based Learning (EBL)

Learning strategies are distinguished by the amount of inference the learner performs on the information provided (Carbonell et al. [1983]). The greater this inference, the smaller is the burden placed on the external environment. EBL (Mitchell et al. [1986]) can be characterized as learning from observation, which is a very general form of inductive learning that includes discovery systems, theory-formation tasks, and the creation of classification criteria to form taxonomic hierarchies. In this type of learning, the learner is not provided with a set of examples of a particular concept; rather, an unsupervised training example is given instead. Besides, the observation may focus on several concepts that need to be acquired, rather than focussing on one concept at a time.

A learning system with the EBL capability utilizes the significance of an event or set of events that results from either whole or part of the other system's planning process and then generalize these events into a new concept (DeJong [1986]). EBL attempts to match the preconditions and the effects of successive actions (operators) and explains the application and sequence of several operators in a given plan in terms of matched variable bindings. The explanation is essentially a proof tree to verify how the goal can be achieved. The plan and the bindings between operators discovered by the EBL process are

retained by the system; it can be retrieved using an indexing scheme when the system is faced with a problem that bears similarities with the one for which the stored plan is a solution.

The following need to be given for carrying out explanation-based learning: (1) Domain Theory, (2) Goal concept; (3) Training example. The domain theory contains knowledge of operators and objects in a representation scheme that the system can manipulate. The initial and goal states are also represented in the same scheme so that the EBL system can recognize the difference between these states and make an operator sequence selection from its domain that will remove this difference.

(1) Domain Theory. This consists of five components:

- A specification of types of resources and their properties
- A specification of types of objects and their properties
- Problem solving operators
- A set of inference rules for inferring properties and relations, including heuristics
- Already known general schemata which are previously learned

The first four components are common to the problem solver in a DSS. Schemata are similar to macro-operators (Fikes et al., [1972]) maintained by other systems, except that they exist in an uninstantiated form, while macro operators tend to be more specific since they are generated by a data driven approach.

(2) Goal Concept. This is a general specification of the goal to be learned. In general, a goal is an incomplete world state yet to be

achieved. It can be specified in non-operational terms without using objects and their properties. But eventually there must be an equivalent goal state description using operational terms.

(3) Training Example. This is an example of the goal concept.

The objective of the EBL process is to determine the proper generalization of the training example that is a sufficient concept definition for the goal concept. If there is no given solution plan for making the decision, the knowledge stored in the domain theory is used to construct one; otherwise the domain theory is used to build a causally complete interpretation of the plan. The plan is an observed sequence of primitive operators of the DSS's problem solver that achieves the goal. In some cases, some of the operators may be missing, then they must be inferred from achieved states given in the input.

There are three processes to achieve an explanatory schema. They are:

- Understand a given example plan for obtaining the solution
- Evaluate the solution plan to see if it is worthwhile converting the plan to a schema
- Generalize the plan to a new schema.

Each of these processes is described in detail below.

Phase 1: Understanding the plan. It is necessary that the system maintain causal relationships justifying every element, including all relevant objects and their properties in the representation. When a

rule is invoked during the explanation phase, a copy of the uninstantiated rule is added to the explanation structure. Unification between specific operators used in the plan and those in the domain theory is made in the specific substitution context. Unification between two operators in the domain theory is made using both the specific and general substitution lists. Any new substitutions required to achieve this are added to these lists. Since a plan consists of operations representing actions (i.e., information-processing activities in a DSS), state information must be included to build dependency links which make up the causal relationships. The links connect each operator in the plan with other operators, existing schemata, and all the inference rules retrieved from domain theory during the understanding process. The understanding process is complete when the causal structure linking the goal and initial states and the two substitution lists are built. An explanation structure is constructed in this phase, resulting in a proof tree modified by replacing each instantiated rule by the associated general rule.

Phase 2: Evaluation. In deciding whether or not to generalize a plan to a new schema, five questions must be answered. These are (DeJong [1983]):

- Is the goal achieved?
- Is the goal a general one?
- Are the resources required available?
- Is the new method of achieving the goal at least as effective as other known ones?
- Does the input match one of the known generalizable patterns?

If the answers to all these questions are in the affirmative, the plan passes the tests for generalization and a new schema is created. This process ensures that all available schemata guarantee optimality in those segments of solution plans where they are employed by the DSS. The actual generalization process (DeJong [1985]) is described below.

Phase 3: Generalization. Assuming the input solution plan is completely understood and a causal structure has been constructed with the accompanying specific and general substitution lists, a generalization of the plan can be performed. The generalization process goes as follows:

Step 1 Eliminate operators and states which do not causally support the goal. The remaining structure is the solution explanation.

Step 2 Identify nominal instantiation of known schemata and eliminate these by retracting the unification between the general specification of the schemata and the specific operators used in the instantiation.

Step 3 Identify operators which support only higher, more general abstractions and eliminate these.

Step 4 If the explanation was constructed from an observed solution, look for sub-goals which can be achieved in a more efficient manner. For each sub-goal in the explanation, check if the system already has a schema for achieving that sub-goal, and insert the more efficient schema.

Step 5 Generate the final generalization by applying the general substitution list to the remaining explanation structure.

The generalized plan in its final form represents the appropriate degree of generalization that was possible from the given plan. The conversion of this plan into a schema involves setting the leaves of the plan (the state represented by the bottom-most input specifications) to be the preconditions of the schema which is given a new name. Indexing the schema takes the form of a rule, say inference rule I99 of the form: IF THE PRECONDITIONS ARE X Y & Z THEN USE SCHEMA#77. If, in the course of generating a subsequent solution plan, the system were to be faced with a set of preconditions X Y & Z, it would accordingly use rule I99 and retrieve SCHEMA#77 which yields an uninstantiated version of the generalized plan created above. This plan is instantiated using problem specific values, and a sub-goal is achieved. We end this section with a DSS example to illustrate the applicationof EBL in DSSs and the potential advantages. The task is to learn to recognize the descriptions for the pair (?L1,?A1) such that the loan L1 with amount A1 would be granted. The ? sign indicates that these variables are uninstantiated. The explanation-based learning problem involves specifiying the following three kinds of information for this example:

Goal Concept:

```
Reasonable-Share-of-Risk(?L1,?A1) & Sufficient-Funds(?L1,?A1)
& Sufficient-Liquid-Assets(?L1,?A1) & Financially-
Profitable(?L1,?A1) & Credit-Rating(High,?L1,?A1)
-> Grant-Loan(?L1,?A1)
```

Training Example:

```
Grant-Loan(LN,ABC)
Industry-Inventories/Curr-Assets(13%)
Inventories/Curr-Assets(40%,ABC)
Cash+Receivables/Curr-Liabilities(55%,ABC)
Industry-Cash+Receivable/Curr-Liabilities(30%)
Industry-Cash-to-Curr-Liabilities(50%)
Cash-to-Curr Liabilities(77%,ABC)
Pro-Forma-Net-Profits/Tangible-Assets(65%,ABC)
Industry-Pro-Forma-Net-Profits/Tangible-Assets(60%)
Industry-Pro-Forma-Pretax-Profits/Tangible-Assets(65%)
Pro-Forma-Pretax-Profits/Tangible-Assets(69%,ABC)
Current-Prime-Rate(7.5%)
Total-Debt($10,000,000,ABC)
Applied-Loan-Amount($1,000,000,ABC)
Funds-from-Operations($50,000,000,ABC)
Industry-Tangible-Net-Worth/Total-Debt(60%)
Tangible-Net-Worth/Total-Debt(89%,ABC)
```

Domain Theory:

```
Rule 1  Reasonable-Share-of-Risk(?a1) & Sufficient Funds(?a1) &
        Sufficient-Liquid-Assets(?a1) &
        Financial-Credit-Rating(High,?a1)
        -> Grant-Loan(?a1)

Rule 2  Tangible-Net-Worth/Total-Debt(?x3,?a3) &
        Industry-Tangible-Net-Worth/Total-Debt(?x4) &
        Greater-than(?x3,?x4)
        -> Reasonable-Share-of-Risk(?a3)

Rule 3  Funds-from-Operations(?x5,?a5) &
        Total-Funds-for-Debts(?x6,?a5) &
        Greater-than(?x5,?x6)
        -> Sufficient-Funds(?a5)

Rule 4  Applied-Loan-Amount(?L7,?a7) &
        Total-Debt (?x7,?a7) &
        Current-Prime-Rate(?r7) &
        Plus(?x8,?L7,?x7) &
        Product(?x9,?x8,?r7)
        -> Total-Funds-for-Debts(?x9,?a7)

Rule 5  Profitability-Rating(High,?a8) &
        Solvency-Rating(High,?a8)
        -> Financial-Credit-Rating(High,?a8)
```

Rule 6
```
Cash-to-Curr-Liabilities(?x10,?a10) &
Industry-Cash-to-Curr-Liabilities(?x11) &
Greater-than(?x10,?x11) &
Cash+Receivables/Curr-Liabilities(?x12,?a10) &
Industry-Cash+Receivables/Curr-Liabilities(?x13) &
Greater-than(?x12,?x13) &
Inventories/Curr-Assets(?x14,?a10) &
Industry-Inventories/Curr-Assets(?x15) &
Greater-than(?x14,?x15)
->Sufficiently-Liquid-Assets(?a10) &
  Solvency-Rating(High,?a10)
```

Rule 7
```
Pro-Forma-Net-Profits/Tangible-Assets(?x16,?a16) &
Industry-Pro-Forma-Net-Profits/Tangible-Assets(?x17) &
Greater-than(?x16,?x17) &
Pro-Form-Pretax-Profits/Tangible-Assets(?x18,?a16) &
Industry-Pro-Forma-Pretax-Profits/Tangible-Assets(?x19) &
Greater-than(?x18,?x19)
 -> Financially-Profitable(?a16) &
    Profitability-Rating(High,?a16)
```

Explanation Structure

Insert Figure 5.1 Here

The unifications used in obtaining the explanation structure are shown in Table 5.1; the general and specific substitution lists (i.e., the G and S lists) are shown in Table 5.2. Finally, the schema learned is shown in Table 5.3. A schema consists of three slots: the antecedents, the concequents, and the application sequence of the rules used in the explanation.

Insert Tables 5.1, 5.2 and 5.3 Here

In the example, the goal concept to be learned, Grant-Loan (?L1,?A1), is defined in terms of the predicates Reasonable-Share-of-Risk, Sufficient Funds, Sufficient-Liquid-Assets, Financially-Profitable, and Credit-Rating. All of these are pretty abstract

descriptions about the goal concept. By contrast, the training example is described in terms of other predicates representing more specific financial characteristics; the firm ABC with these descriptions is granted a loan. The domain theory includes a set of rules and facts that allow explaining how the training example is a member of the goal concept. The task of explanation-based learning is to learn to describe the goal concept with more operational terms, as used in the training example, so that it can be achieved by following the set of applied rules. The product of the explanation-based learning process is the explanation structure and the schema.

The schema in Table 5.3 defining the goal concept in terms of the predicates describing the training example (i.e., the observation). These descriptions are operational because they can be obtained through retrieving financial data of the firm and simple arithmetic calculation. The schemata generated by explanation-based learning can be very useful for decision-support since they help accelerate the problem-solving process by aggregating the most relevant pieces of knowledge in a logical way. This is achieved by learning with a rich set of domain knowledge.

For the inductive learning methods discussed in Section 3, they are based on searching for features common to the training examples. These methods, referred to as similarity-based learning, are primarily used for generating classification rules, heuristics, or concept descriptions. By contrast, EBL is based on a rich set of knowledge of the task domain to search an explanation (i.e., a proof) to justify the given observation. The explanation can then be generalized into

a structured schema to facilitate the decision process for similar problems. The EBL problem is part of the Learning Module described in Section 4.

6. Conclusions

In this paper, we have presented a learning augmented approach to the design of intelligent DSSs by adding a Learning and Knowledge Acquisition Unit. The Learning and Knowledge Acquisition Unit can acquire decision rules through an inductive learning engine; it can also refine the rules or derive decision schemata by four functional components: the Instance Selector, the Problem Solver, the Critic, and the Learning Module. This learning augmented methodology provides a unified framework for supporting such important DSS operations as knowledge acquisition, rule learning and refinement, improving model manipulation, and deriving heuristics for model selection. To achieve these learning functions, we have employed machine learning techniques such as similarity-based learning, learning by experimentation, and explanation-based learning.

Acknowledgements

This research is supported in part by the Office for Information Management, the Prochnow Foundation, the Research Board, and the Amoco Foundation Professorship. Thanks are due to Professors Gerald DeJong, R. Michalski and Larry Randell for making available their machine learning systems. The author would also like to thank S. Park, U. Menon, and S. Piramuthu, the doctoral students working in this project, for their assistance. Piramuthu ran the empirical study for

comparing inductive learning methods (Piramuthu and Shaw [1988]). The hardware and software equipments used in this research are donated by IntelliCorp and Texas Instruments. Finally, the author is grateful to Professors J. Gentry and D. Whitford for providing the data from their bankruptcy study.

References

Applegate, L. M., Konsynski, B. R., and Nunamaker, J. F., 1986, "Model Management Systems: Design for Decision Support," Decision Support Systems, Vol. 2, pp. 81-91.

Blanning, R., 1986, "An Entity-Relationship Approach to Model Management," Decision Support Systems, Vol. 2, No. 1, pp. 65-72.

Bonczek, R. H., Holsapple, C. W. and Whinston, A. B., 1981, "Representing Modeling Knowledge with First Order Predicate Calculus," Operations Research.

Bonczek, R. H., Holsapple, C. W. and Whinston, A. B., 1981, Foundations of Decision Support Systems, Academic Press, New York.

Bonczek, R. H., Holsapple, C. W. and Whinston, A. B., 1983, "Specification of Modeling and Knowledge in Decision Support Systems," in H. G. Sol (ed.), Processes and Tools for Decision Support, (North Holland: Amsterdam).

Buchanan, G. B. and Mitchell, T. M., 1978, "Model-Directed Learning of Production Rules," Pattern-Directed Inference Systems, in Waterman, D., and Hayes-Roth, F. (eds.) (New York: Academic Press).

Bundy, A., Silver, B., and Plummer, D., 1985, "An Analytical Comparison of Some Rule-Learning Programs," Artificial Intelligence, 27, pp. 137-181.

Carbonell, 1983, "Learning by Analogy: Formulating and Generalizing Plans from Past Experiences," in Machine Learning, ed. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Tioga.

Davis, R., 1979, "Interactive Transfer of Expertise: Acquisition of New Inference Rules," Artificial Intelligence, Vol. 12, pp. 121-57.

DeJong, G., 1986, "An Approach to Learning from Observation," in Machine Learning (Vol. II), Michalski et al. (Eds.), (Morgan Kaufmann, Los Altos, CA).

Dietterich, T. G. and Michalski, R. S., 1983, "A Comparative Review of Selected Methods for Learning from Examples," in Machine Learning: An AI Approach, ed. R S. Michalski, J. G. Carbonell, and T. M. Mitchell, Tioga.

Dolk, D. and Konsynski, B., 1984, "Knowledge Representation for Model Management Systems," IEEE Trans. on Software Engineering, Vol. SE-10, No. 4, pp. 619-627.

Dutta, A. and Basu, A., 1984, "An Artificial Intelligence Approach to Model Management in Decision Support Systems," IEEE Computer, Vol. 17, No. 9, pp. 89-98.

Elam, J. J. and Henderson, J. C., 1980, "Knowledge Engineering Concepts for Decision Support System Design and Implementation," Proceedings of Fourteenth Annual Hawaii International Conference on System Sciences, (Western Periodicals: North Hollywood, CA).

Elam, J. and Konsynski, B., 1987, "Using Artificial Intelligence Techniques to Enhance the Capabilities of model Management Systems," Decision Sciences, Vol. 18, pp. 487-502.

Fedorowicz, J. and Williams, G. B., 1986, "Representing Modeling Knowledge in an Intelligent Decision Suport System," Decision Support Systems 2, pp. 3-14.

Fikes, R., Hart, P., and Nilsson, N., 1972, "Learning and Executing Generalized Robot Plans," Artificial Intelligence, Vol. 3, No. 4, pp. 251-288.

Gentry, J., Newbold, P., and Whitford, D., 1985, "Classifying Bankrupt Firms with Funds Flow Components," Journal of Accounting Research, Vol. 23, No. 1, pp. 146-160.

Henderson, J., 1987, "Finding Synergy Between Decision Support Systems and Expert Systems Research," Decision Sciences, Vol. 18, pp. 333-349.

Holland, J., 1986, "Escaping Brittleness: The Possibility of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems," Machine Learning: An AI Approach, Michalski, Carbonell, and Mitchell (Eds.), Tioga Pub. Co., Palo Alto, CA.

Konsynski, B. and Sprague, R. H., Jr., 1986, "Future Research Directions in Model Management," Decision Support Systems, 2, pp. 103-109.

Korf, R., 1985, Learning to Solve Problems by Searching for Macro-operators, (Pitman, Marshfield, Mass.).

Langley, P., 1981, "Data-Driven Discovery of Physical Laws," Cognitive Science, Vol. 5, pp. 31-54.

Langley, P., 1984, "Learning to Search: From Weak Methods to Domain-Specific Heuristics," CMU-RI-TR-84-21, (The Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA), also appeared in Cognitive Science (1986).

Lee, W. D., and Ray, S., 1986, "Rule Refinement Using the Probabilistic Rule Generator," Proceedings of the Fifth National Conference on Artificial Intelligence, pp. 442-447.

Lenat, D. B., 1983, "EURISKO: A Program that Learns New Heuristics and Domain Concepts; The Nature of Heuristics III: Program Design and Results," Artificial Intelligence, Vol. 21, pp. 61-98.

Liang, T. P., 1987, "Development of a Knowledge-Based Model Management System," BEBR Paper No. 1364, Department of Accountancy, University of Illinois.

Menon, U., and Shaw, M., 1988, "Qualitative Reasoning in Intelligent Decision Support Systems," Technical Report, Department of Business Administration, UIUC, Champaign, IL (a Progress Report to the Prochnow Foundation).

Michalski, R. S., 1980, "Pattern Recognition as Rule-Guided Inductive Inference," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 2, pp. 249-361.

Michalski, R. S., 1983, "A Theory and Methodology of Inductive Learning," in Michalski, R., Carbonell, J., and Mitchell, T., (eds.), Machine Learning, Tioga Publishing Co., Palo Alto, CA.

Mitchell, T., 1982, "Generalization As Search," Artificial Intelligence, Vol. 18, pp. 203-226.

Mitchell, T., Utgoff, P. E. and Banerji, R., 1983, "Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristic," in Machine Learning: An Artifical Intelligence Approach, R S. Michalski, J. G. Carbonell, and T. M. Mitchell (eds.), Tioga Publishing Co., Palo Alto, CA.

Mitchell, T., Keller, R., and Kedar-Cabelli, S., 1986, "Explanation-based Generalization: A Unifying View," Machine Learning, 1, pp. 47-80.

O'Rorke, P., 1982, "A Comparative Study of Inductive Learning Systems AQ11P and ID-3 Using a Chess Endgame Test Problem," Report No. UIUCDCS-F-82-899, Department of Computer Science, University of Illinois at Urbana-Champaign.

Piramuthu, S., and Shaw, M., 1988, "A Comparative Study on Inductive Learning Systems," Technical Report, Department of Business Administration, UIUC, Champaign, IL.

Quinlan, J. R., 1979, "Discovering Rules from Large Collection of Examples: A Case Study," in Michie, D. (ed.) Expert Systems in the Micro Electronic Age, (University Press: Edenburgh).

Quinlan, J. R., 1986, "The Effects of Noise on Concept Learning," in Machine Learning: An Artificial Intelligence Approach, Michalski et al., (Eds.), Tiogo, Los Altos, pp. 149-165.

Rendell, L., 1983, "A New Basis for State-Space Learning Systems and a Successful Implementation," Artificial Intelligence, Vol. 20, pp. 369-92.

Rendell, L., 1986, "A General Framework for Induction and a Study of Selective Induction," Machine Learning, Vol. 1, No. 2, pp. 177-226.

Rendell, L., Benedict, P., Cho, H., and Seshu, R., 1987, "Improving the Design of Rule Learning Systems," Report No. UIUCDCS-R-87-1395, Department of Computer Science, University of Illinois at Urbana-Champaign.

Samuel, A. L., 1967, "Some Studies in Machine Learning Using the Game of Checkers II--Recent Progress," IBM J. of Research and Development, Vol. 11, No. 6, pp. 609-617.

Shaw, M. and Gentry, J., 1988, "Incorporating Inductive Learning in A Knowledge-Based DSS for Evaluating Business Loans," Journal of Financial Management, Fall, 1988.

Shaw, M., Menon, U., Park, S., 1988, "Applying Explanation-Based Learning to Automated Process Planning," to appear in Expert Systems for Manufacturing Design, A. Kusiak (Ed.), Society of Manufacturing Engineers, Deerborn, MI.

Shaw, M., 1987, "Applying Inductive Learning to Enhance Knowledge-based Expert Systems," Decision Support Systems, Vol. 3, No. 4, pp. 319-332.

Sprague, R. H. and Carson, E. D. 1982, Building Effective Decision Support Systems, (Prentice-Hall Inc.: Englewood Cliffs, NJ).

Stohr, E. A., 1986, "Decision Support and Knowledge-based System: A Special Issue," Journal of Management Information Systems, Vol. II, No. 4.

Winston, P., 1984, Artificial Intelligence, (Addison-Wesley: Reading, MA).

Appendix 1.

RULE077 [PROFITABILITYRULES]
If 1) firm's 3-year average net profits is greater than 0, and
2) industry median ratio of pretax profits to total tangible assets divided by prime rate of interest is greater than or equal to 1, and
3) firm's ratio of pretax profits to total tangible assets divided by prime rate of interest is greater than or equal to 1, and
4) firm's percentile in industry ratio of pretax profits to total tangible assets is greater than or equal to .75, and
5) firm;s percentile in industry ratio of net profits to tangible net worth is greater than or equal to .75, and
6) firm's percentile in industry inventory turnover rate is greater than or equal to .5,
Then the firm's profitability rating is HIGH.

```
PREMISE:  ($AND (GREATERP* (VAL1 CNTXT P1) 0)
                (GREATEQ* (VAL1 CNTXT R1) 1)
                (GREATEQ* (VAL1 CNTXT R2) 1)
                (GREATEQ* (VAL1 CNTXT R3) .75)
                (GREATEQ* (VAL1 CNTXT R4) .75)
                (GREATEQ* (VAL1 CNTXT R5) .5))
ACTION:   (DO-ALL
                (CONCLUDE CNTXT PROFITABILITY-RATING HIGH TALLY 1000))
```

RULE020 [EVALUATIONRULES]
If 1) The credit-worthiness measure, S1, is known, and
2) the indication of the extent to which a customer relationship with the firm, S2, will build the bank is known, and
3) the evaluation of expected profitability to the bank of a customer relationship with the firm, S3, is known, and
4) the weight which the bank's management gives to the credit-worthiness S1 is known, and
5) the weight which the bank's management gives to build the bank S2 is known, and
6) the weight which the bank's management gives to the profitability S3 is known,
Then the final evaluation score is [[[S1 times the weight which the bank's management gives to the credit-worthiness S1] plus [the indication of the extent to which a customer relationship with the firm will build the bank times the weight which the bank's management gives to build the bank S2]] plus [the evaluation of expected profitability to the bank of a customer relationship with the firm times the weight which the bank's management gives to the profitability S3]].

```
PREMISE:  ($AND (KNOWN CNTXT S1)  (KNOWN CNTXT S2)
                (KNOWN CNTXT S3) (KNOWN CNTXT W1)
                (KNOWN CNTXT W2) (KNOWN CNTXT W3))
```

```
ACTION:  (DO-ALL
           (CONCLUDE CNTXT FINAL-EVAL-SCORE
             (PLUS
               (PLUS
                 (TIMES (VAL1 CNTXT S1)  (VAL1 CNTXT W1))
                 (TIMES (VAL1 CNTXT S2)  (VAL1 CNTXT W2)))
               (TIMES (VAL1 CNTXT S3) (VAL1 CNTXT W3)))
           TALLY 1000))


RULE073 [S1RULES]
-------
If 1) 1) a rating of management competence is HIGH, and
      2) the outside credit rating of the firm is HIGH, and
      3) credit rating besed on the bank's financial
         analysis of teh applicant's statements is HIGH, or
   2) 1) a rating of management competence is HIGH, and
      2) the outside credit rating of the firm is HIGH, and
      3) credit rating besed on the bank's financial
         analysis of teh applicant's statements is AVERAGE, or
   3) 1) a rating of management competence is HIGH, and
      2) the outside credit rating of the firm is GOOD, and
      3) credit rating besed on the bank's financial
         analysis of teh applicant's statements is HIGH, or
   4) 1) a rating of management competence is AVERAGE, and
      2) the outside credit rating of the firm is HIGH, and
      3) credit rating besed on the bank's financial
         analysis of teh applicant's statements is HIGH,
Then 1) it is definite (100%) that S1 is 3, and
     2) it is definite (100%) that extensive credit check is
        finished, and
     3) Inform the user of this decision.

PREMISE:  ($AND
            ($OR
              ($AND (SAME CNTXT F1 HIGH)
                (SAME CNTXT F2 HIGH) (SAME CNTXT F3 HIGH))
              ($AND (SAME CNTXT F1 HIGH)
                (SAME CNTXT F2 HIGH)
                (SAME CNTXT F3 AVERAGE))
              ($AND (SAME CNTXT F1 HIGH)
                (SAME CNTXT F2 GOOD) (SAME CNTXT F3 HIGH))
              ($AND (SAME CNTXT F1 AVERAGE)
                (SAME CNTXT F2 HIGH) (SAME CNTXT F3 HIGH))))
ACTION:  (DO-ALL (CONCLUDE CNTXT S1 3 TALLY 1000)
           (CONCLUDE CNTXT EXTENSIVE-CREDIT-CHECK YES TALLY
           1000)
           (SPRINTT "After an evaluation of the firm's credit-worthines
            in relation to a particular loan application, the
            overall credit rating, is high."))
```

Appendix 2

Production Rules for Applying Models in Loan Evaluation

In a DSS, production rules can be used to represent model knowledge. The application of each model is directed by an if-then rule and interpreted as "if the input requirements are satisfied and the model thus becomes executable, then the output value is. . . ." In the model predicates, we use the upper case to specify the model, underlines to represent the input values, and the rest to represent the output values. Some of the rules directing model applications in a loan-evaluation DSS are listed here. Machine learning techniques can be used to learn additional rules or to refine existing rules.

1. (var1,?x1,yr1,fn) & (var2,?x2,yr1,fn) & (REGRESS,?x1,?x2,?x3,?x4,yr,fn) => (β,var1,var2,?x3,yr,fn) & (R^2,var1,var2,?x4,yr,fn)

   With the input values, ?x1 and ?x2, of var1 and var2 in a given year for a particular firm, the REGRESS model outputs values, ?x3 and ?x4, of β and R^2 between the two input variables.

2. (var1,?x1,yr,fn) & (var2,?x2,yr,fn) & (RATIO,?x1,?x2,?x3,yr,fn) => (ratio,var1,var2,?x3,yr,fn)

   Using the input values, ?x1 and ?x2, of var1 and var2 in a given year for a given firm, the Ratio model calculates the value of their ratio, ?x3.

3. (var,?x1,yr,fn) & (var,?x2,(-yr1),fn) & (var,?x3,(-yr2),fn) & (AVG,?x1,?x2,?x3,?x4,yr,fn) => (avg,var,?x4,yr,fn)

   Using the input values, ?x1,?x2, and ?x3, of var from three consequentive years, the AVERAGE model calculates their average value, ?x4.

4. (var,?x1,yr,fn) & (industry-type,?x2,yr,fn) & (PERCENTILE,?x1,?x2,?x3,yr,fn) => (percentile,var,?x3,yr,fn,?x2)

   Using the value of var and the industry type of this firm, the PERCENTILE model calculates its percentile value of var in its industry.

5. (var,?x1,yr,fn) & (industry-type,?x2,yr,fn) & (MEDIAN,?x1,?x2,?x3,yr,fn) = > (median,var,?x3,yr,fn,?x2)

   Using the value of var and the industry type of this firm, the MEDIAN model calculates its median value of var in its industry.

6. (var,?x1,yr,fn) & (tax-type,?x2,yr,fn) & (TAX,?x1,?x2,?x3,yr,fn) = > (after-tax,var,?x3,yr,fn)

   Using the value of var and the tax-type of this firm, the TAX model calculates the after-tax value of var.

7. (var,?x1,yr,fn) & (var,?x2,(-,yr,1),fn) & (var,x3,(-,yr,2),fn) & (TREND,?x1,?x2,?x3,?x4,yr,fn) = > (trend,var,?x4,yr,fn)

   Using the value of var form three consequentively years, the TREND model calculates the trend of var.

8. (ratio,(+,long-term-debt,curr-liab,?x1,yr,fn),total-assets,?x2,yr, fn) & (ratio,funds-from-op,(+,interest,(avg,debt-maturity,?x4,yr, fn) &(trend,sales,?x5,yr,fn) & (RISK-SCOPE,?x1,?x2,?x3,?x4,?x5, ?x6,yr,fn) = > (risk-score,?x6,yr,fn)

   Using (long-term-debt + current-liabilities) to total-assets ratio, and funds-from-operation to (interest + the-average-debt-maturity) ratio, the RISK-SCORE model calculates the risk score of this firm.

9. (interest-income,?x1,yr,fn) & (cost-of-handling-deposit,?x2,yr,fn) & (avg,loan-volume,?x3,yr,fn) & (avg,collected-balance,?x4,yr,fn) & (risk-score,?x5,yr,fn) & (LT,?x5,0) & (LOAN-YIELD-I,?x1,?x2,?x3, ?x4,?x5,?x6,yr,fn) = > (loan-yield,?x6,yr,fn)

   Using the interest-income, cost-of-handling-deposit, three year average loan-volume and collected-balance, the risk-score, under the condition that the risk score is less than 0, the LOAN-YIELD-I model calculates the loan-yield of this firm.

10. (interest-income,?x1,yr,fn) & (cost-of-handling-deposit,?x2,yr,fn) & (avg,loan-volume,?x3,yr,fn) & (avg,collected-balance,?x4,yr,fn) & (risk-score,?x5,yr,fn) & (GT,?x5,0) & (LOAN-YIELD-II,?x1,?x2, ?x3,?x4,?x5,?x6,yr,fn) = > (loan-yield,?x6,yr,fn)

    Using the interest-income, cost-of-handling-deposit, three year average loan-volume and collected-balance, and the risk-score, under the condition that the risk score is greater than 0, the LOAN-YIELD-II model calculates the loan-yield of this firm.

11. (interst-income,?x1,yr,fn) & (cost-of-handling-deposit,?x2,yr,fn) & (avg,loan-volume,?x3,yr,fn) & (avg,collected-balance,?x4,yr,fn) & (risk-score,?x5,yr,fn) & (GT,?x5,1.255) & (LOAN-YIELD-III,?x1, ?x2,?x3,?x4,?x5,?x6,yr,fn) = > (loan-yield,?x6,yr,fn)

    Using the interest-income, cost-of-handling-deposit, three year average loan-volume and collected-balance, and the risk-score, under the condition that the risk score is greater than 1.255, the LOAN-YIELD-III model calculates the loan-yield of this firm.

12. (interest-income,?x1,yr,fn) & (cost-of-handling-deposit,?x2,yr,fn) & (avg,loan-volume,?x3,yr,fn) & (avg,collected-balance,?x4,yr,fn) & (risk-score,?x5,yr,fn) & (GT,?x5,2.79) & (LOAN-YIELD-IV,?x2,?x2, ?x3,?x4,?x5,?x6,yr,fn) = > (loan-yield,?x6,yr,fn)

    Using the interest-income, cost-of-handling-deposit, three year average loan-volume and collected-balance, and the risk-score, under the condition that the risk score is greater than 2.79, the LOAN-YIELD-IV model calculates the loan-yield of this firm.

13. (trend,interest-rate,?x1,yr,fn) & (interest-rate,?x2,yr,fn) & (loan-period,?x3,yr,fn) & (GT,?x3,3,12) & (ST-LOAN-RATE,?x1,?x2, ?x3,?x4,yr,fn) = > (st-loan-rate,?x4,yr,fn)

    Using the trend of interest-rate, interest-rate, and the loan-period, under the condition that the loan-period is between 3 to 12 months, the ST-LOAN-RATE model calculates the short term loan rate of this firm.

14. (trend,interst-rate,?x1,yr,fn) & (interest-rate,?x2,yr,fn) & (loan-period,?x3,yr,fn) & (GT,?x3,12) & (LT-LOAN-RATE,?x1,?x2,?x3, ?x4,yr,fn) = > (lt-loan-rate,?x4,yr,fn)

    Using the trend of interest-rate, interest-rate, and the loan-period, under the condition that the loan-period is greater than 12 months, the LT-LOAN-RATE model calculates the long term loan rate of this firm.

15. (interest-cost,?x1,yr,fn) & (operating-cost,?x2,yr,fn) & (avg-assets,?x3,yr,fn) & (reserve-requirements,?x4,yr,fn) & (COST-OF-FUNDS,?x1,?x2,?x3,?x4,?x5,yr,fn) = > (cost-of-funds,?x5,yr,fn)

    Using the interest-cost, operating-cost, three year average assets, and the reserve-requirements, the COST-OF-FUND model calculates the cost-of-fund of this firm.

16. (cost-of-funds,?x1,yr,fn) & (loan-yield,?x2,yr,fn) & (COMPENSATING-BALANCE,?x1,?x2,?x3,yr,fn) = > (compensating-balance,?x3,yr,fn)

    Using the cost-of-funds, and the loan-yield, the COMPENSATING-BALANCE model calculates the compensating-balance of this firm.

Appendix 3 An Example Illustrating the Schema Refinement Process.

This appendix describes an example applying the learning method described in Section 4 to refine an existing model manipulation schema. As described in Section 4, every schema contains a condition part which describes a class of applicable problems and a solution part which displays the shared solution plan for every problem in this class. The G and S sets are kept for each schema for further refinement. The initial schema is shown in Figure A-1, with G and S defined for the version space. The generalization relations between the domain variables are organized into a hierarchy shown in Figure A-2.

In Figure A-1, a model manipulation schema is created from an initial positive instance, (percentile, (ratio, A/R, inv), ?x1, 1986, ABC), which represents the computation modules for getting the percentile value of the ratio between accounts-receivable (A/R) and inventory (inv) in a given year (1986) for a particular firm (ABC). This initial schema has a version space where the G set is the maximally generalization of this instance as, (percentile, (ratio, var1, var2), ?x1, yr, fn), base on the generalization hierarchy shown in Figure A-2. The S set is initiated to be the training instance. In Figure A-3, the schema is applied to a new instance: (percentile (ratio, asset, liab), ?x1, 1986, ABC). Since the instantiated solution tree is complete, the instance is classified as a positive example. It modifies the current version space by minimally generalizing the S set. Based on Figure A-2, asset is the minimal generalization of asset and accounts-receivable, and B/S-var is the minimal generalization of liability and inventory. Therefore, minimally generalizing S

would result in, (percentile, (ratio, asset, B/S-var), ?x1, 1986, ABC).

In Figure A-4, the training instance, (percentile, (ratio, profits, assets), ?x1, 1988), has an incomplete solution tree. Consequently, this instance is classified as a negative example for the current schema. It then modifies the current version space by constraining the G set to be (percentile, (ratio, B/S-var, var2), yr, fn), (percentile, (ratio, var1, I/S-var), yr, fn), or (percentile, (ratio, var1, var2), yr, fn) (yr < 1988).

In Figure A-5, a near-miss example modifies the schema by adding one more precondition of the RATIO model, (avg, var1, ?x5, yr, fn). The G and S sets in the current version space are also updated to include the maximal and minimal generalizations of this example.

Table 5.1

Unification and Resulting S and G Lists

_____ Specific Unification Only
═══ Both Specific and General Unifications

Unification	Specific (piecewise)	General (incremental)
U1	Goal(ABC/?a1)	
U2	Goal(ABC/?a8)	?a8/?a1
U3	Goal(ABC/?a10)	?a10/?a8, ?a10/?a1
U4	13%/?x15	
U5	40%/?x14, ABC/?a10, ABC/?a8, ABC/?a1	
U6	55%/?x12	
U7	30%/?x13	
U8	50%/?x11	
U9	77%/?x10	
U10	Goal(ABC/?a16)	?a16/?a8, ?a16/?a10, ?a16/?a1
U11	65%/?x16, ABC/?a16	
U12	60%/?x17	
U13	65%/?x19	
U14	69%/?x18	
U15	Goal(ABC/?a5)	?a5/?a1, ?a5/?a16, ?a5/?18, ?a5/?a10
U16	Goal(ABC/?a7)	?a9/?x6, ?a7/?a5, ?a7/?a1, ?a7/?a16, ?a7/?a8, ?a7/?a10
U17	7.5%/?r7	
U18	$10,000,000/?x7, ABC/?a7	
U19	$1,000,000/?L7, $11,000,000/?x8, $825,000/?x9, $825,000/?x6	
U20	$50,000,000/?x5, ABC/?a5	
U21	Goal(ABC/?a3)	?a3/?a1, ?x9/?x6, ?a3/?a7, ?a3/?a5, ?a3/?x16, ?a3/?a8, ?a3/?a10
U22	60%/?x4	
U23	89%/?x3, ABC/?a3	

Table 5.3

Schema for Granting Loan to the Company Subject to the Applied Loan Amount

```
Schema:
    :ants
    ;;; ratio related
    Cash-to-Curr-Liabilities(?x10,?a3) &
    Industry-Cash-to-Curr-Liabilities(?x11) &
      Greater-than(?x10,?x11) &
    Cash+Receivables/Curr-Liabilities(?x12,?a3) &
    Industry-Cash+Receivables/Curr-Liabilities(?x13) &
      Greater-than(?x12,?x13) &
    Inventories/Curr-Assets(?x14,?a1) &
    Industry-Inventories/Curr-Assets(?x15) &
      Greater-than(?x14,?x15) &
    Pro-Forma-Pretax-Profits/Tangible-Assets(?x18,?a3) &
    Industry-Pro-Forma-Pretax-Profits/Tangible-Assets(?x19) &
      Greater-than(?x18,?x19) &
    Pro-Forma-Net-Profits/Tangible-Assets(?x16,?a3) &
    Industry-Pro-Forma-Net-Profits/Tangible-Assets (?x17) &
      Greater-than(?x16,?x17) &
    Tangible-Net-Worth/Total-Debt(?x3,?a3) &
    Industry-Tangible-Net-Worth/Total-Debt(?x4) &
      Greater-than(?x3,?x4)
    ;;; loan amount related
    Applied-Loan-Amount(?L7,?a3) &
    Total-Debt(?x7,?a3) &
    Current-Prime-Rate(?r7) &
      Plus(?x8,?L7,?x7) &
      Product(?x9,?x8,?r7)
    Funds-from-Operations(?x5,?a3) &
      Greater-than(?x5,?x9)

    :cons
    Grant-Loan(?a3)

    :ruleseq
    (Rule 2 -> Rule 4 -> Rule 3 -> Rule 7 -> Rule 6 -> Rule 5 -> Rule 1)
```

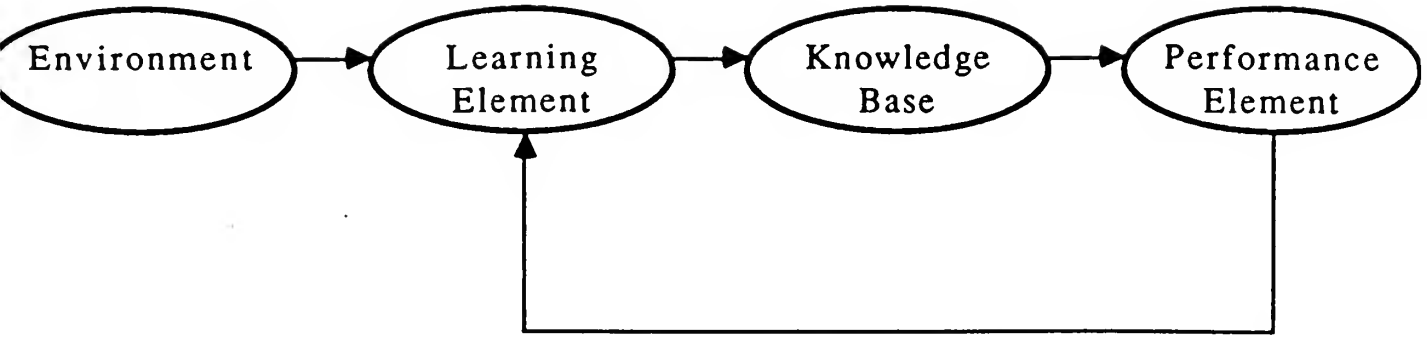


Figure 2.1 The Basic Model of a Machine Learning System

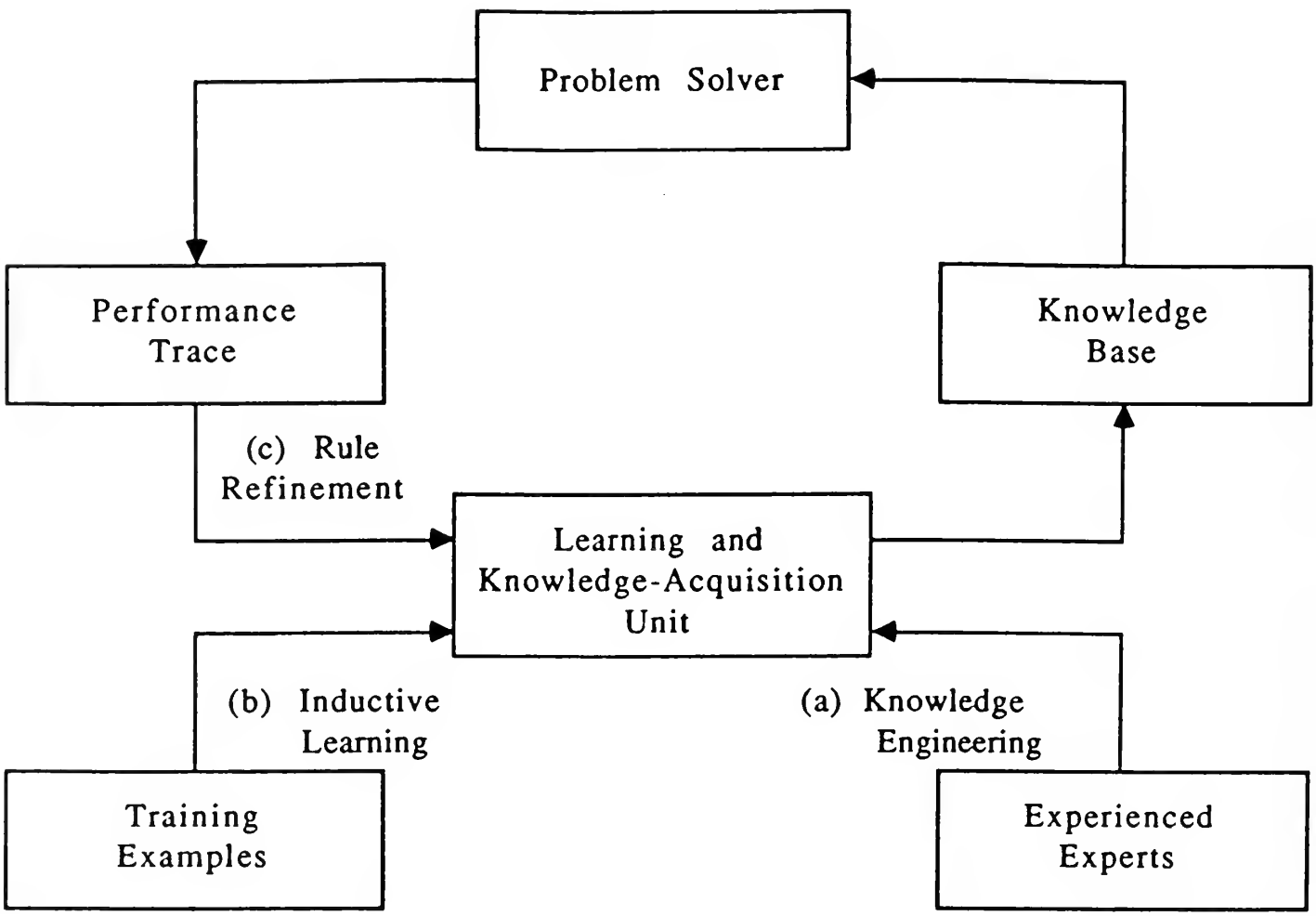


Figure 2.2 Interactions Between the Learning Module and Other DSS Components

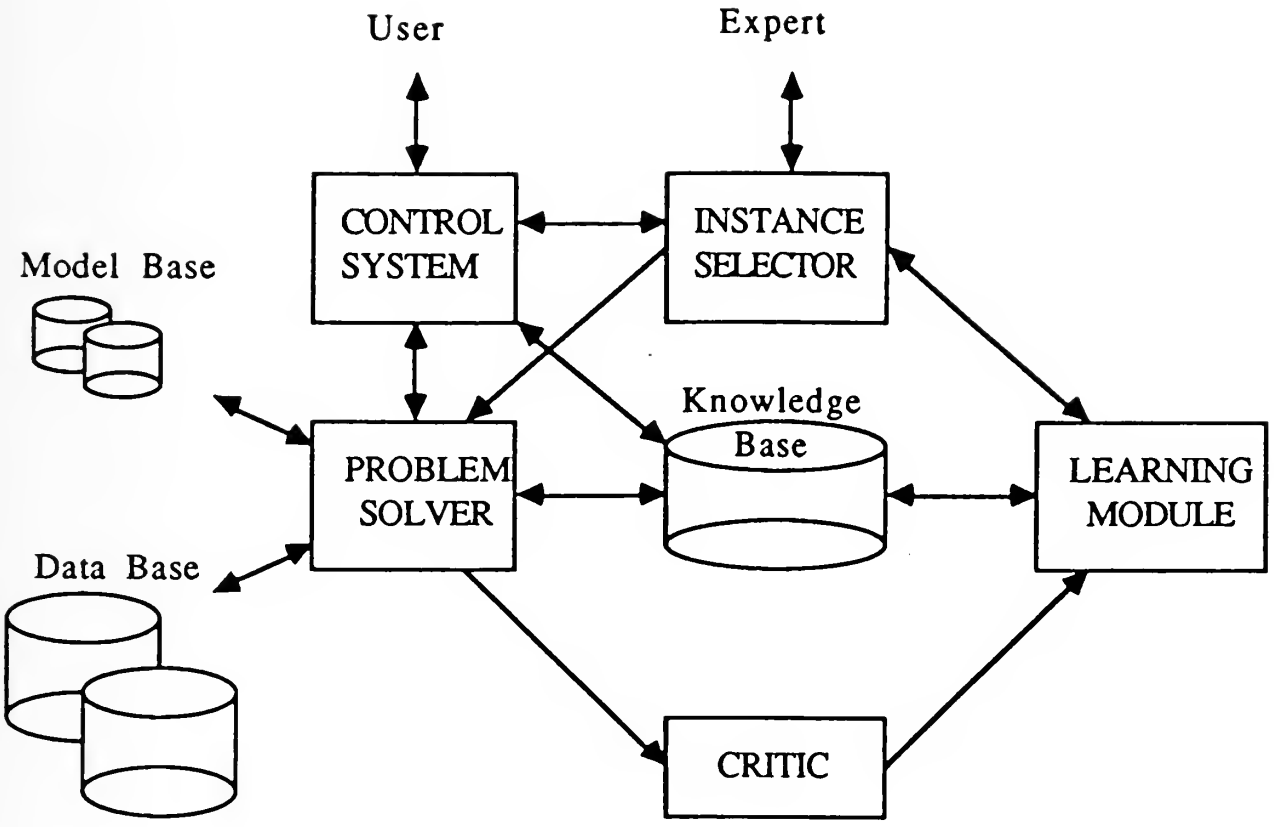


Figure 2.3 The Learning-Augmented DSS Framework for Knowledge Refinement

Classification	I			IA			II		
Company Code	A	B	C	D	E	F	G	H	I
Mgmt-rating	H	H	H	A	H	A	A	M	A
Credit-rating	H	H	A	A	A	A	M	A	A
Current assets	57	39	43	42	38	52	45	37	46
Net-worth	57	55	49	37	46	40	38	29	36
Total-debt	23	17	20	19	28	25	36	27	35
Funds	9	8	7	8	9	6	-9	7	5
Cash	4	3	5	6	4	5	6	6	5
Cur. liability	39	28	47	55	39	45	57	53	57
Inventory	21	15	18	12	14	11	7	13	14
Avg-inventory	9	14	11	6	6	5	3	5	6
Avg-profits	12	15	13	8	9	9	9	9	-0.8
Past-acc-eval	1Y	2Y	3Y	2Y	1Y	1Y	3Y	2Y	NA
Cust-status	C	C	N	C	C	N	N	C	C
Account-type	C	E	D	D	T	E	E	T	T

Legend: H=High, A=Average, M=Medium
C=Current, N=New
C=Commission, E=Employee-trade, D=Deposits, T=Trust-funds

Table 3.1 A Classification Example

CORRECT CLASSIFICATION (%) USING
9 ATTRIBUTES X 58 OBSERVATIONS

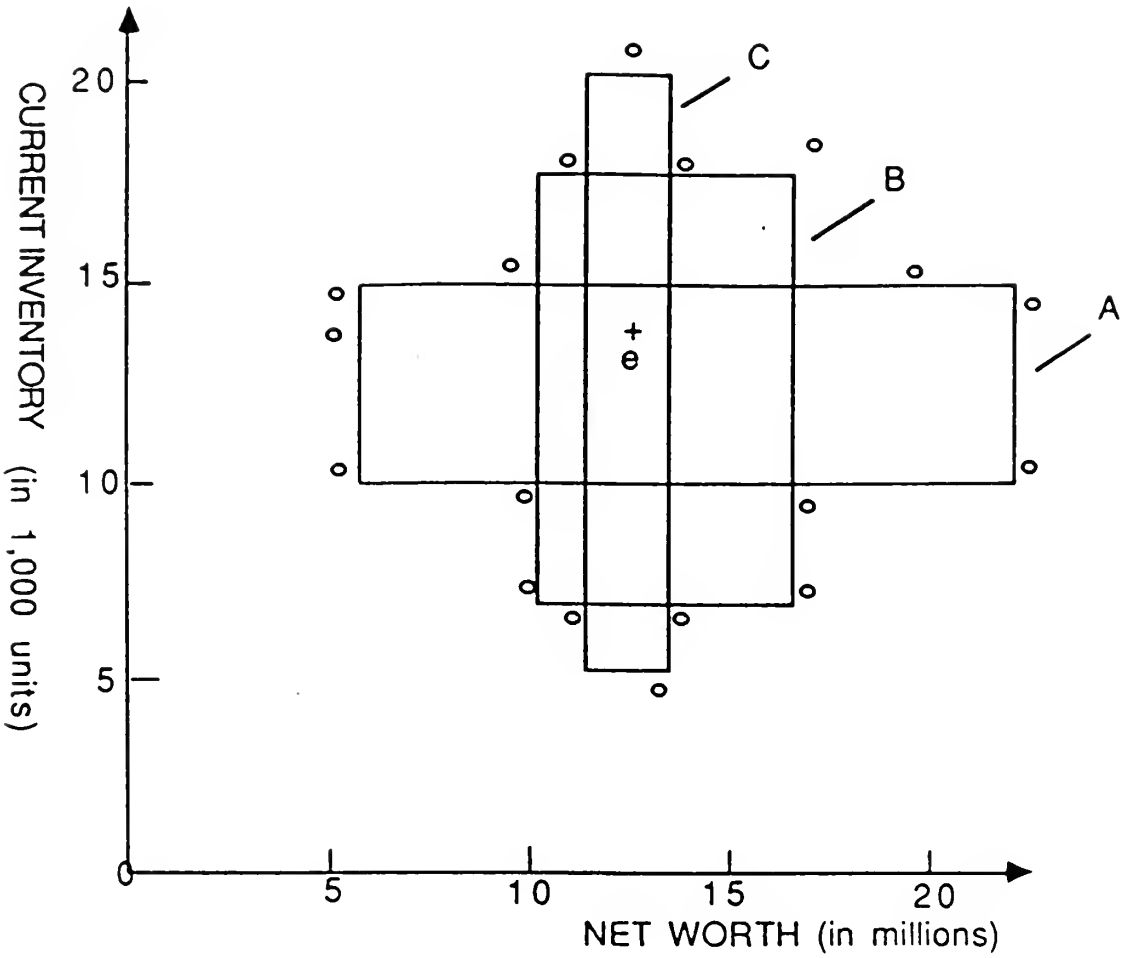
	CLASS A	CLASS B	TOTAL
ACLS	73	73	73
AQ-15	64	82	73
PLS	82	82	82

Using the testing data only

	CLASS A	CLASS B	TOTAL
ACLS	89.7	89.7	89.7
AQ-15	86.2	93.1	89.7
PLS	93.1	93.1	93.1

Using both the testing and the training data sets

Table 3.2 The Classification Results



+ = The positive example e

o = Negative Examples in E

A = [5 < NET-WORTH < 23][10 < CURRENT-INVENTORY < 15]

B = [10 < NET-WORTH < 17][7 < CURRENT-INVENTORY < 18]

C = [11 < NET-WORTH < 14][5 < CURRENT-INVENTORY < 20]

Figure 3.1 An Example of the Star G (e/E)

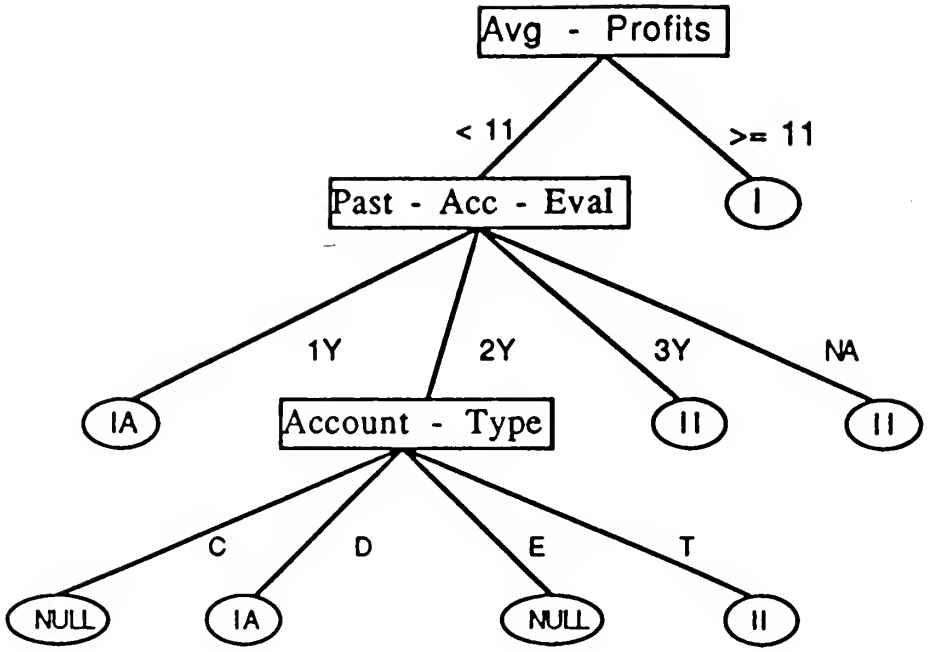


Figure 3.2 The Induction Tree Generated by ID3

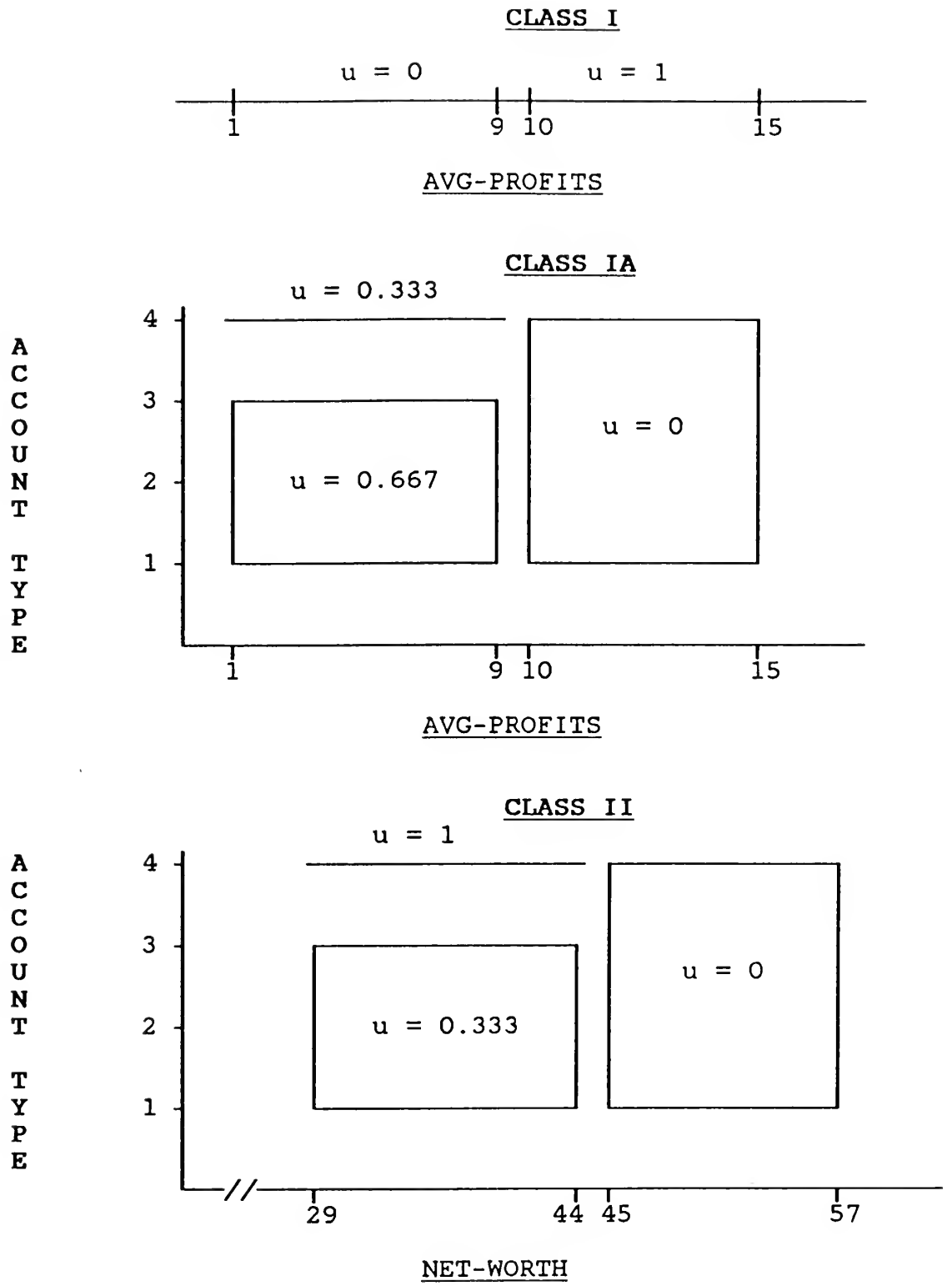


Figure 3.3 The Regions Generated by PLS for Concept Descriptions

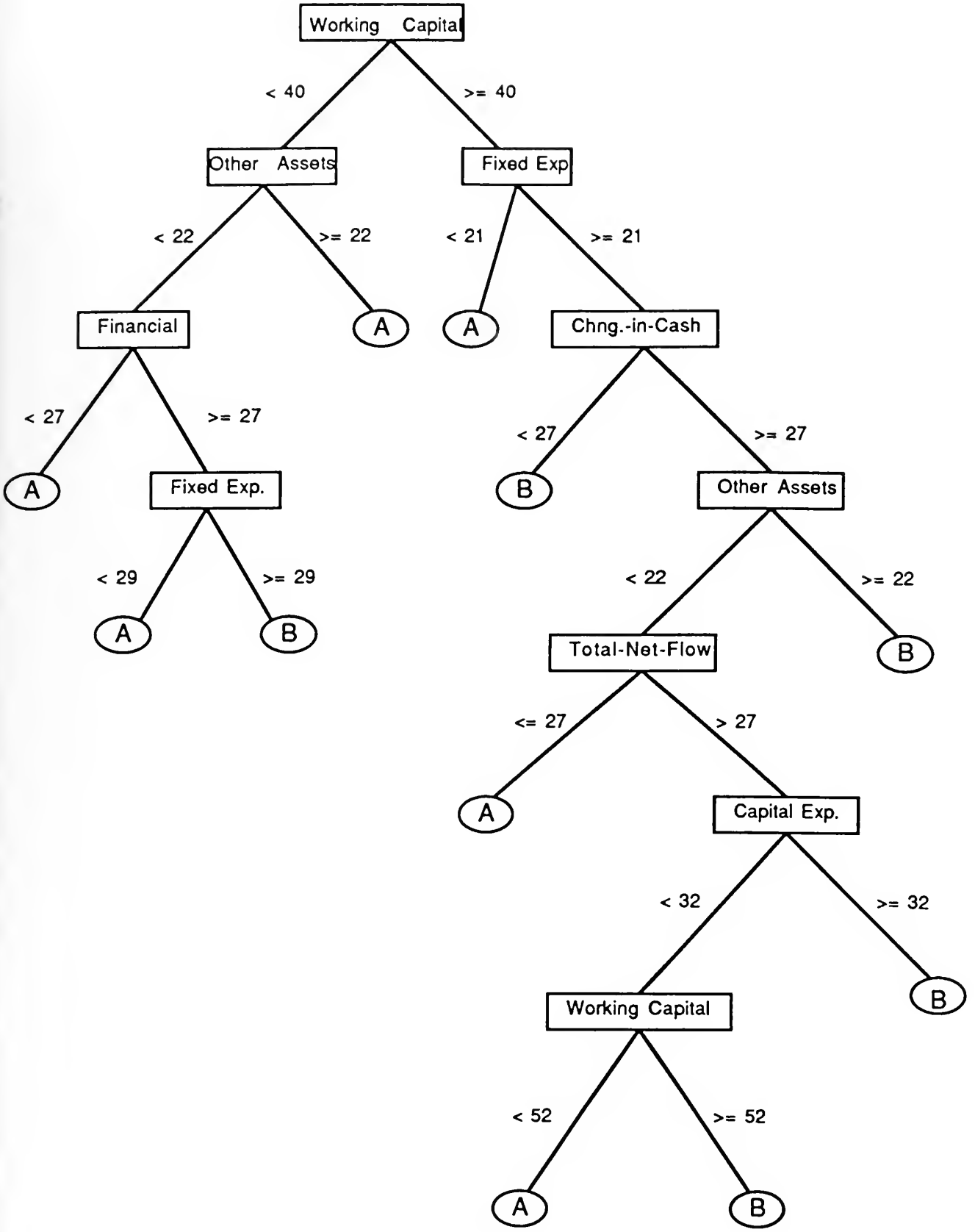


Figure 3.4 The Induction Tree for Classifying the Bankruptcy Data

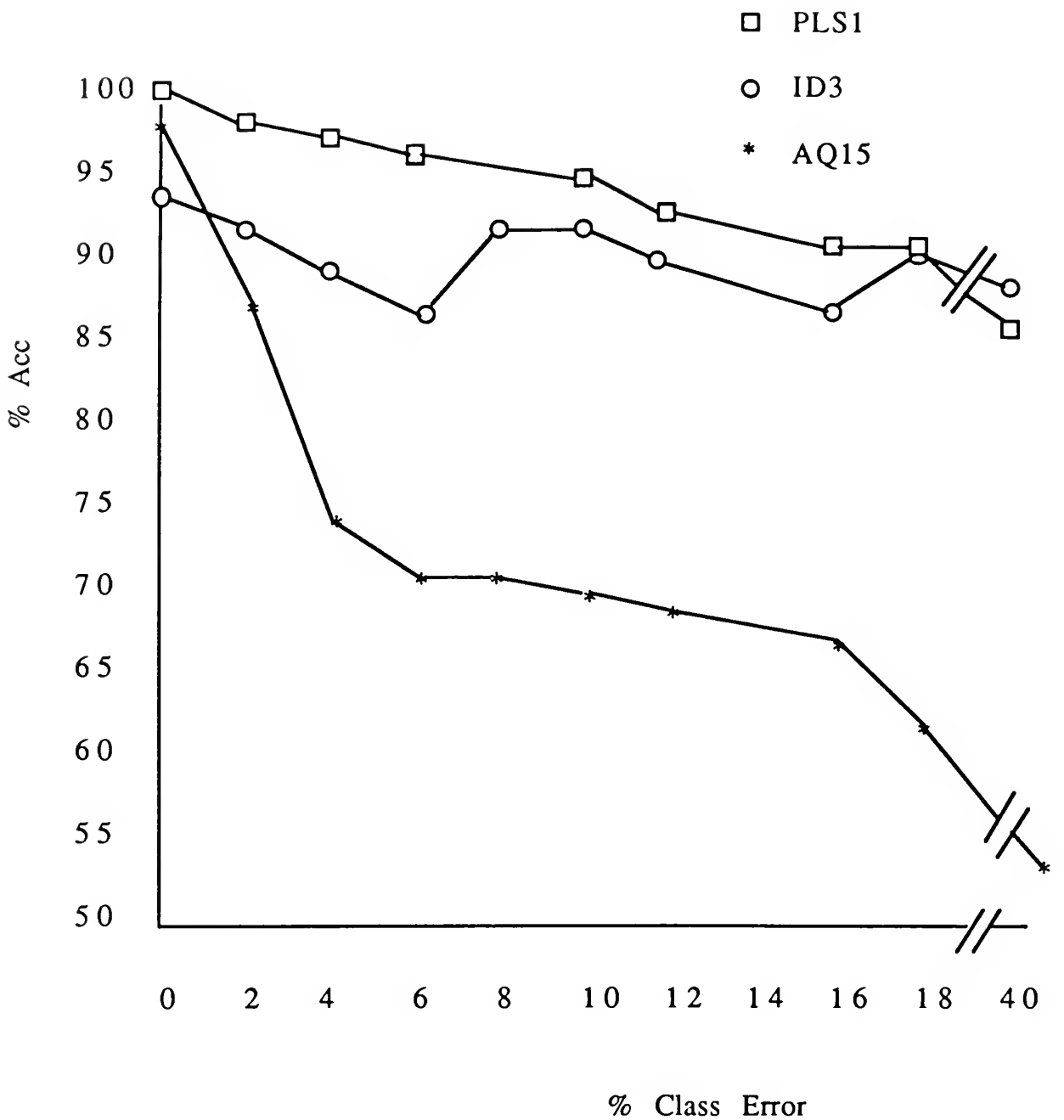


Figure 3.5 (a) Variation of Concept Accuracy with Class Error for Three Systems

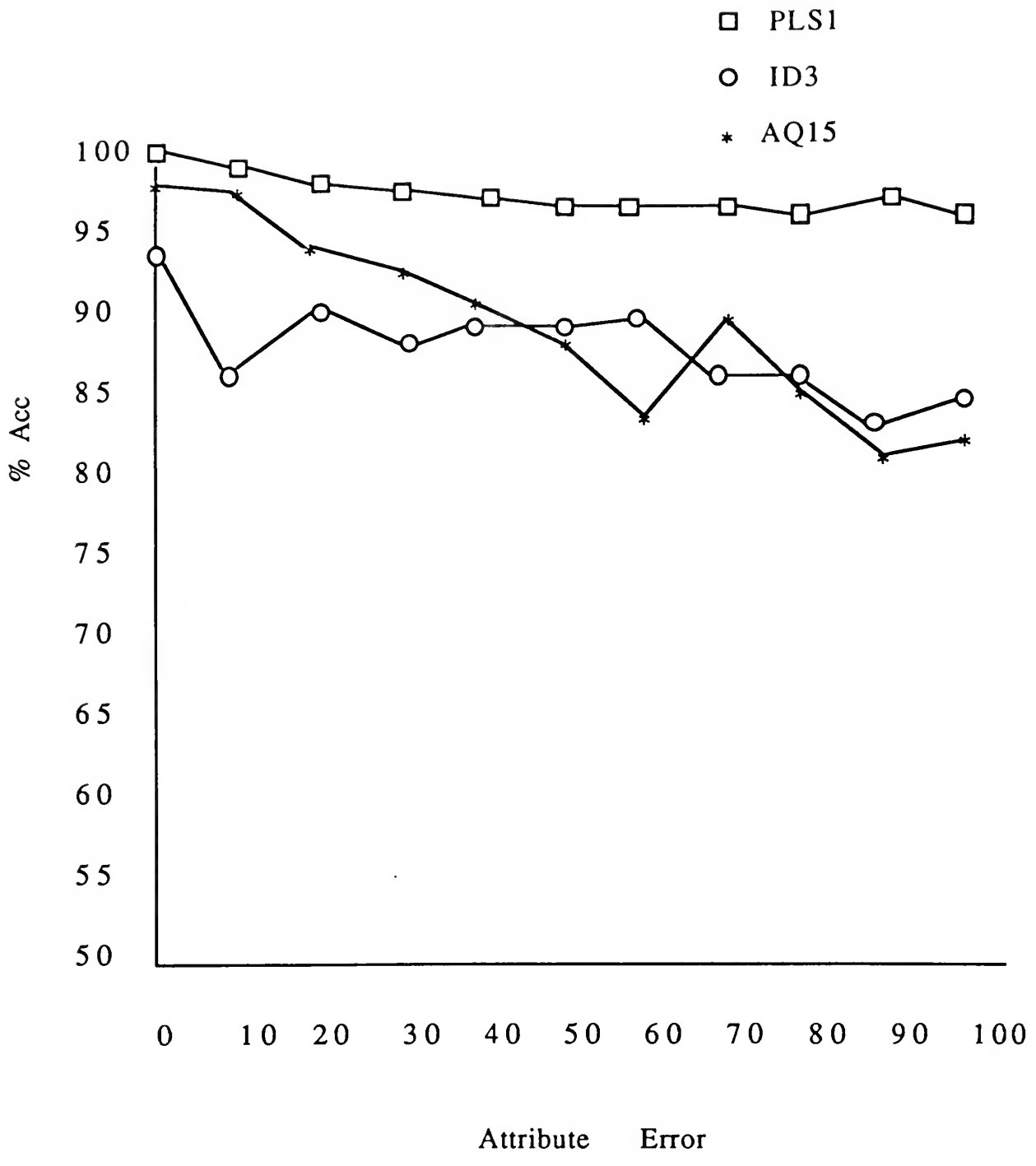


Figure 3.5 (b) Variation of Concept Accuracy with Attribute Error for Three Systems

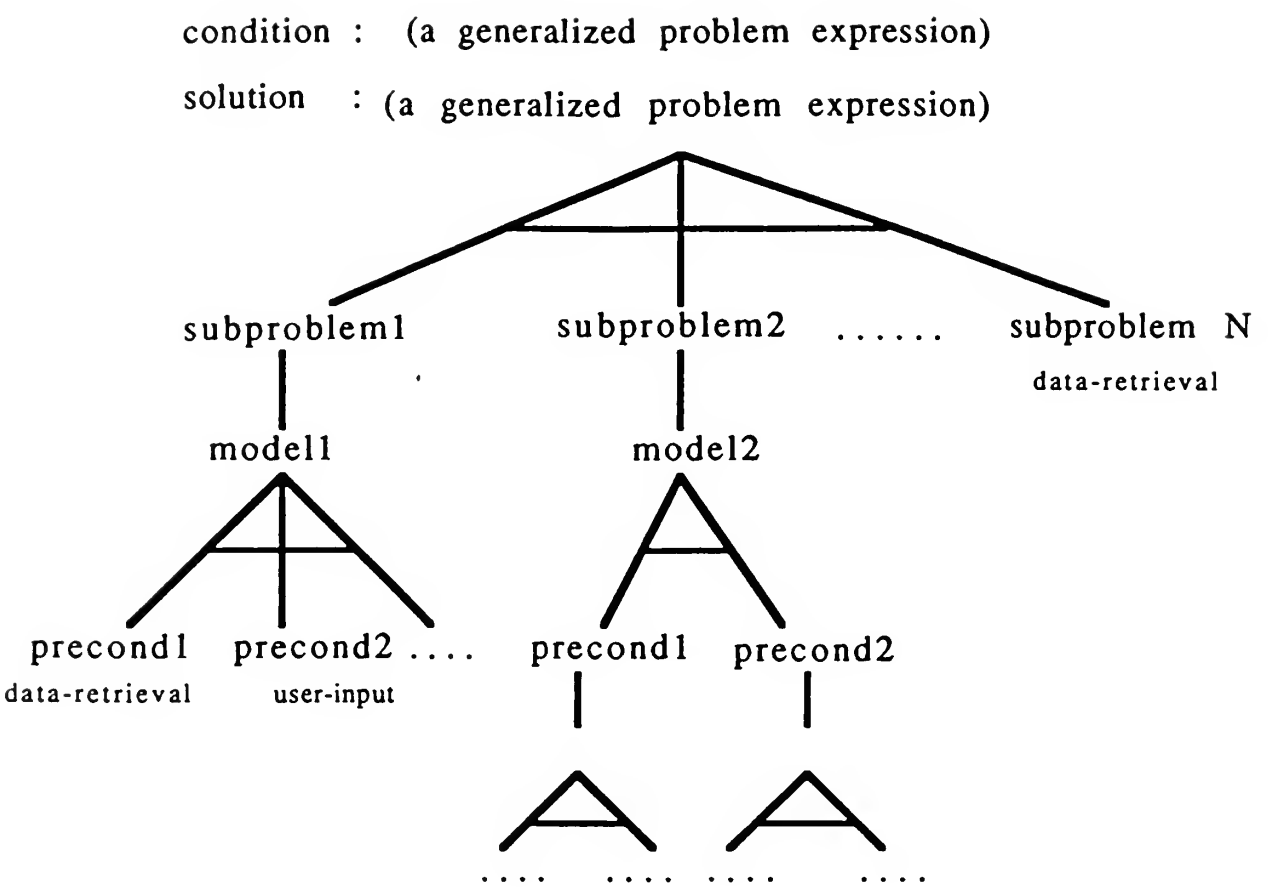
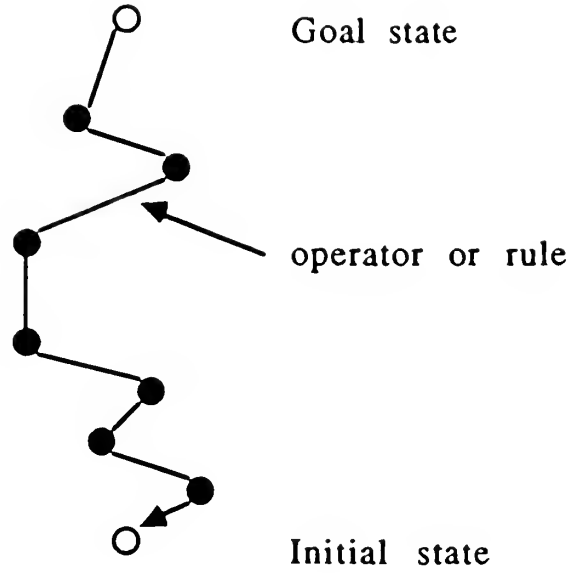
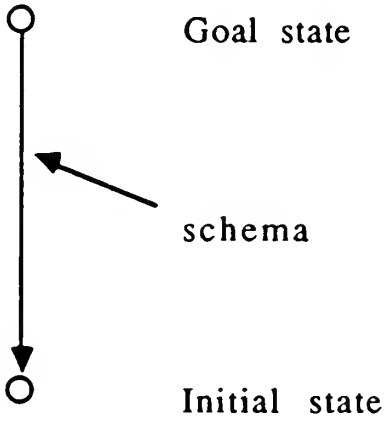


Figure 4.1 The Application of A Model Manipulation Schema



(a)



(b)

Figure 4.2 Search Processes (a) Without and (b) With Schemata

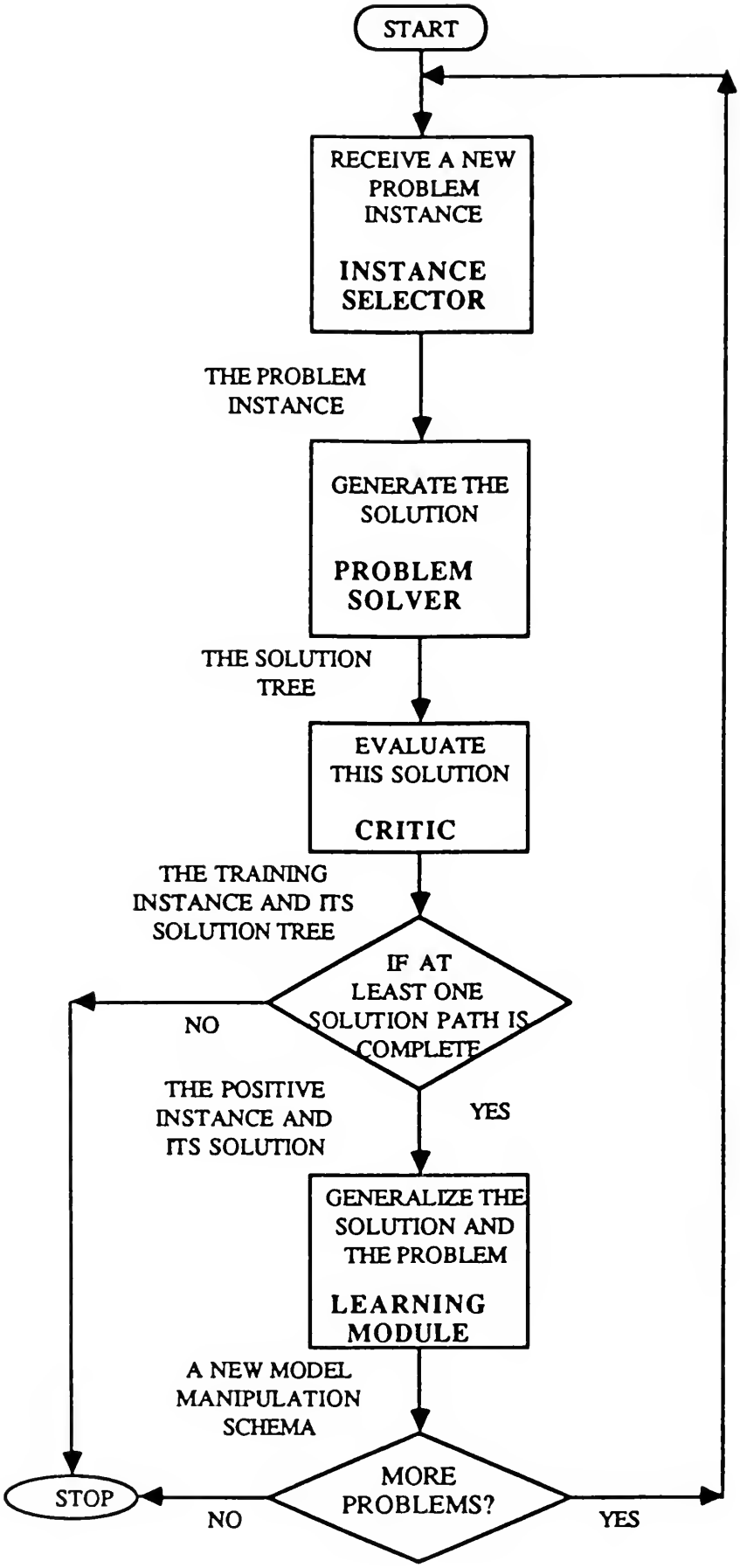


Figure 4.3 The Learning Procedure in the Acquisition of Model Manipulation Knowledge

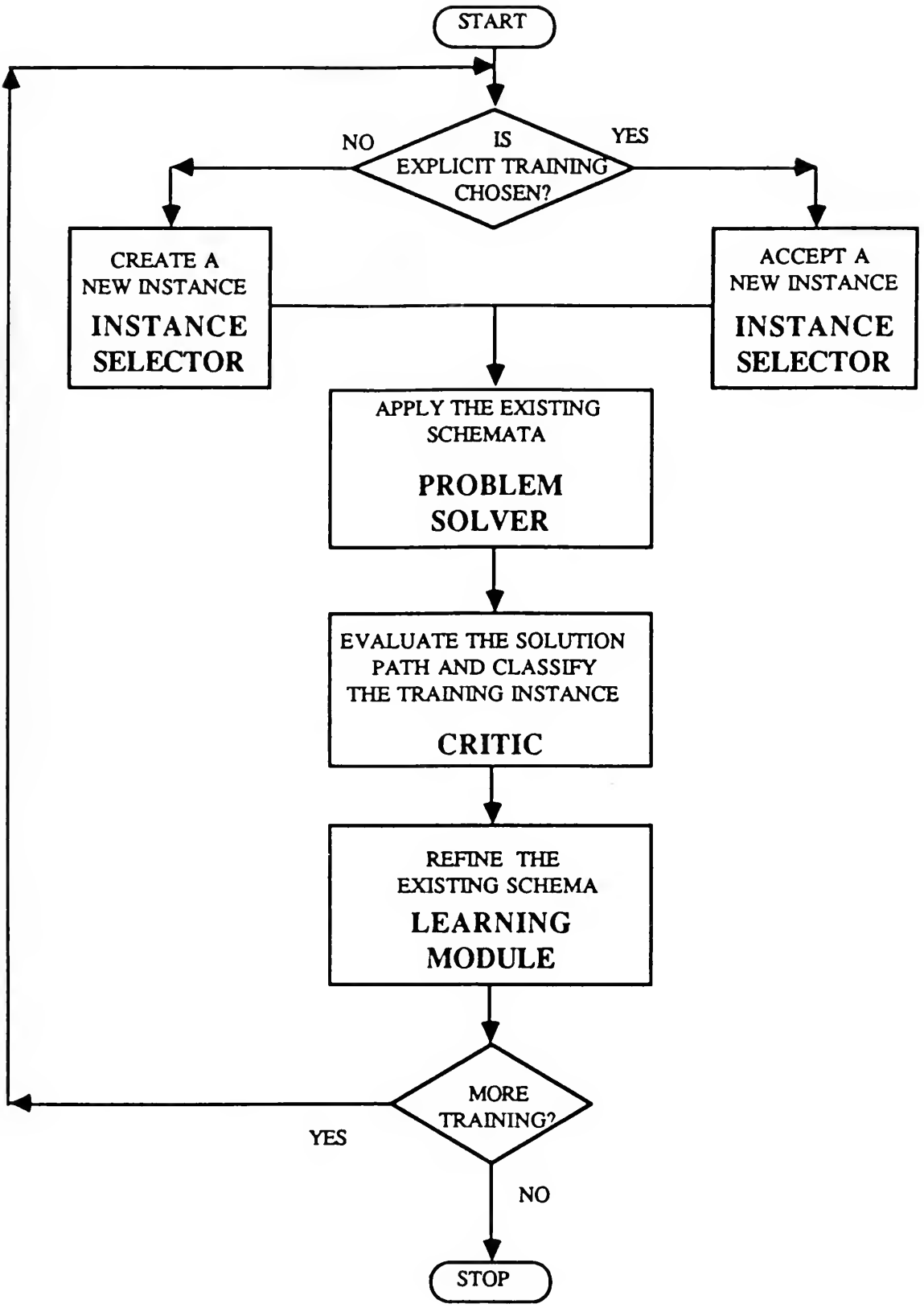
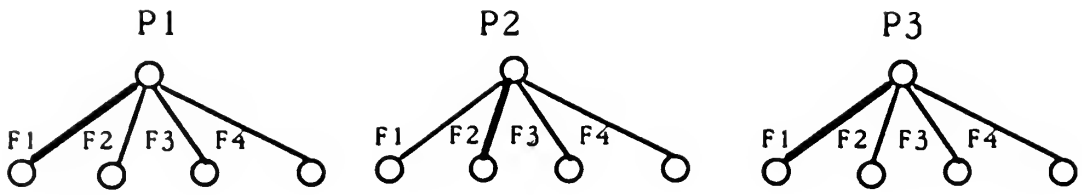


Figure 4.4 The Learning Procedure in the Refinement of Model Manipulation Knowledge

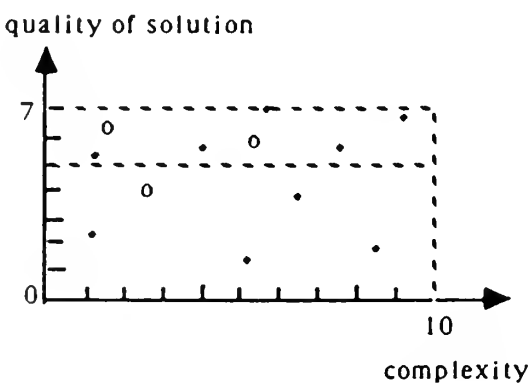
(a)



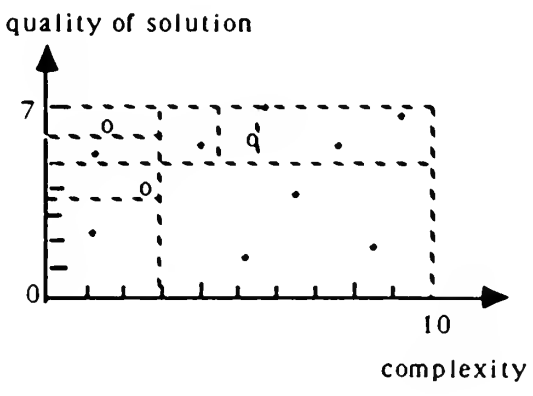
F1: REGRESSION
F2: MOVING AVERAGE
F3: EXPONENTIAL SMOOTHING
F4: DELPHI

P1: THE SALES NEXT YEAR
P2: THE INVENTORY THREE YEARS LATEI
P3: THE INTEREST EXPENSE NEXT YEAR

(b)



(c)



o -- denotes a positive instance
• -- denotes a negative instance

Figure 4.5 An Example of Learning Model Selection Heuristics Using the PLS Approach

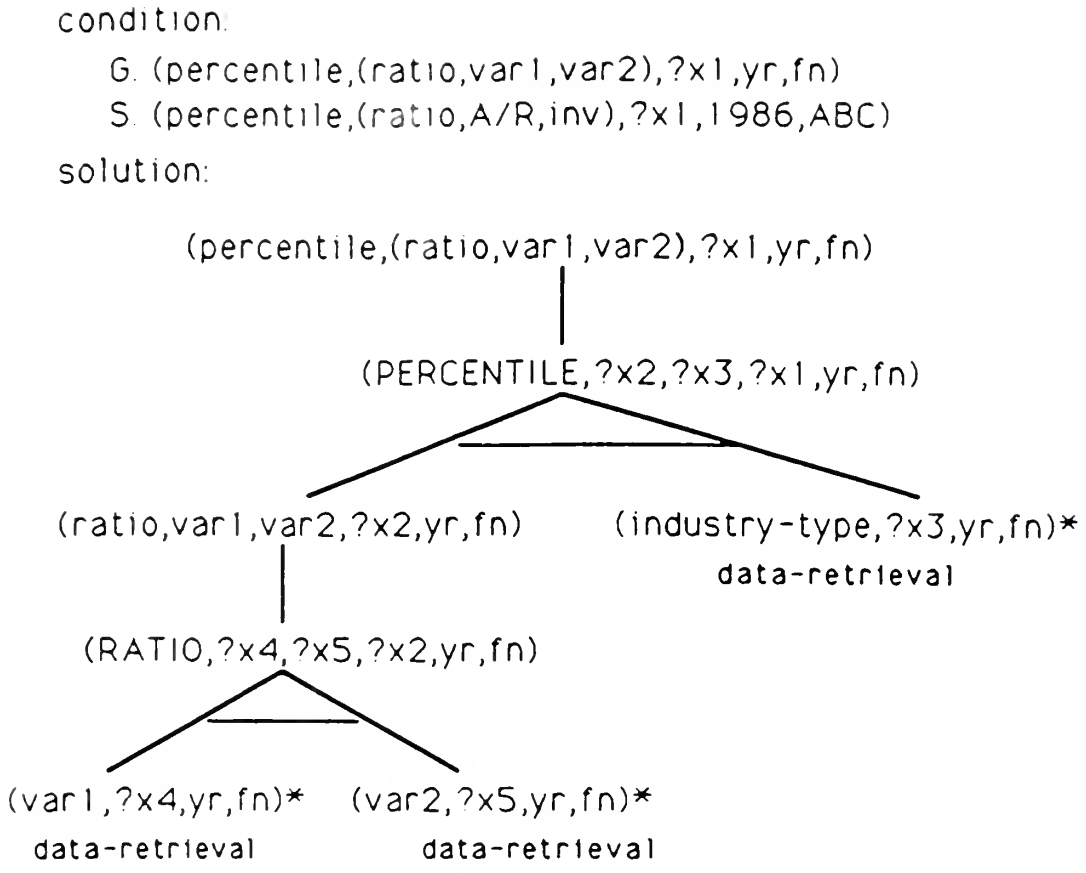
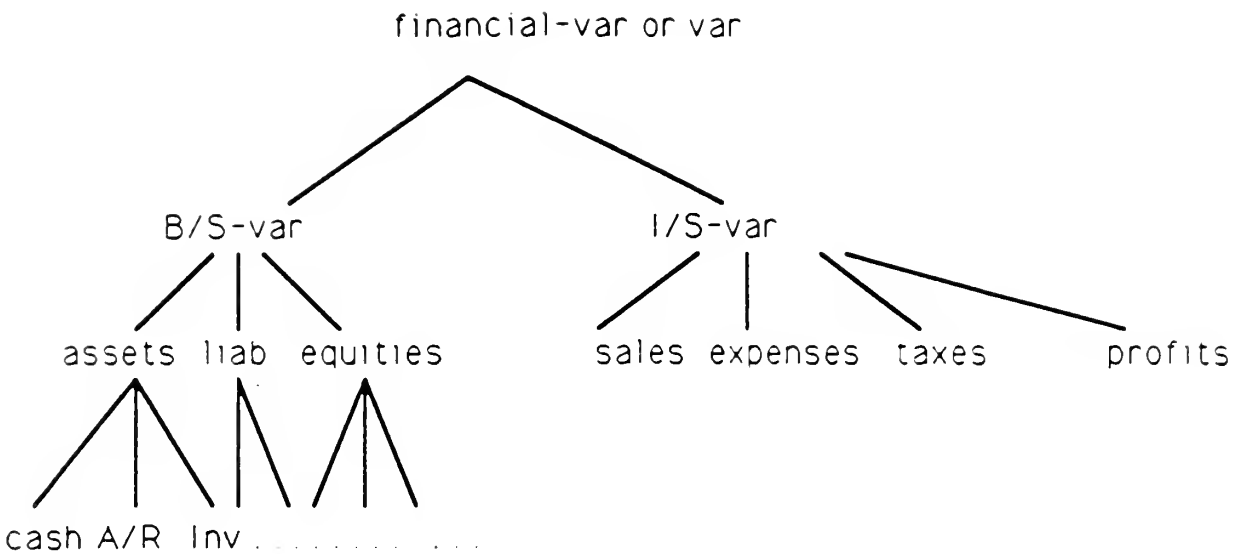


Figure A-1. The Initial Schema

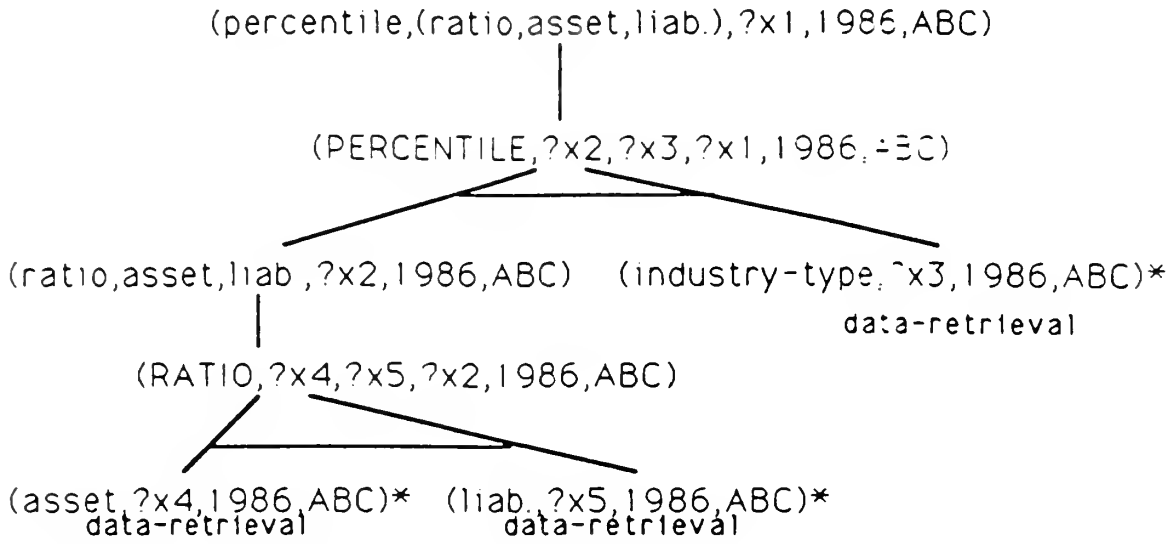


Legend

B/S-var	Balance-sheet-variable
I/S-var	Income-statement-variable
A/R	Accounts-receivable
Inv	Inventory

Figure A-2. The Generalization Hierarchy

- An example,(percentile,(ratio,asset,liab.),?x1,1986,ABC) with the pattern-matching, (assets/var1,liab/var2,1986/yr,ABC/fn), has the following instantiated solution:



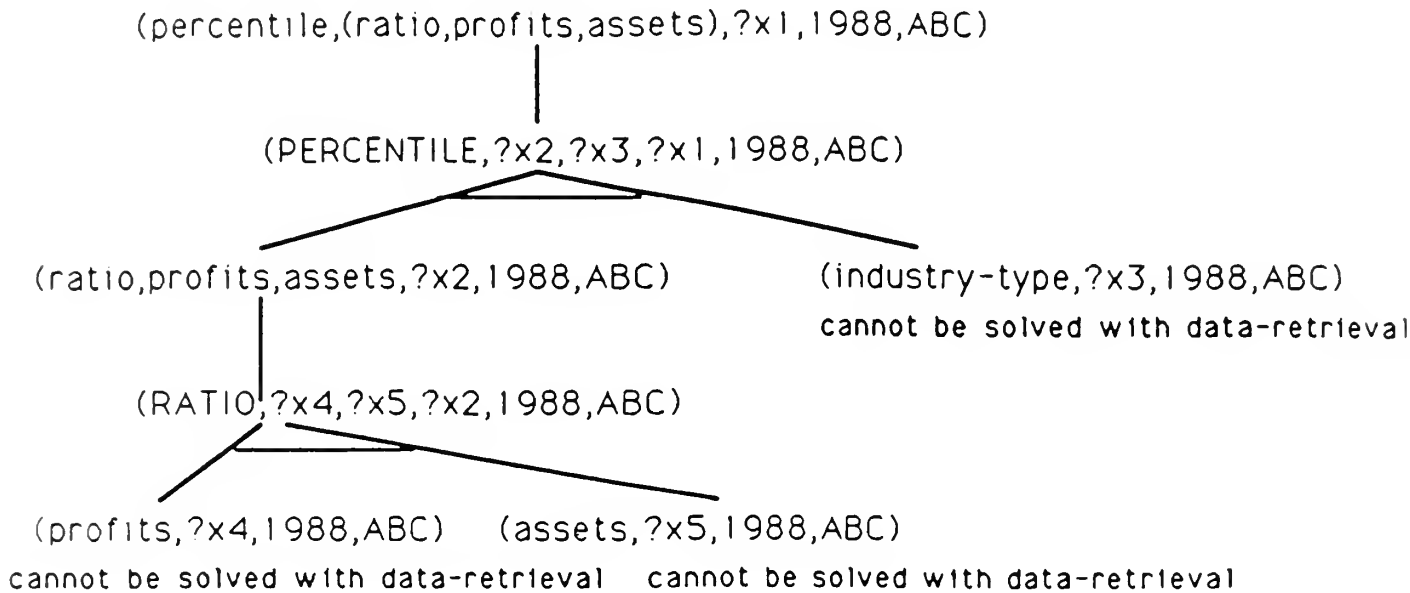
- This example is classified as a positive example, since all terminal nodes are solvable,. It modifies the condition of this schema as follows:

  G: (percentile,(ratio,var1,var2),?x1,yr,fn)

  S (percentile,(ratio,asset,B/S-var),?x1,1986,ABC)

Figure A-3. Applying the Schema to a Positive Example

- An example, (percenitle,(ratio,profits,assets),?x1,1988,ABC), with the pattern-matching, (profits/var1,assets/var2,1988/yr,ABC/fn) has the following instantiated solution:

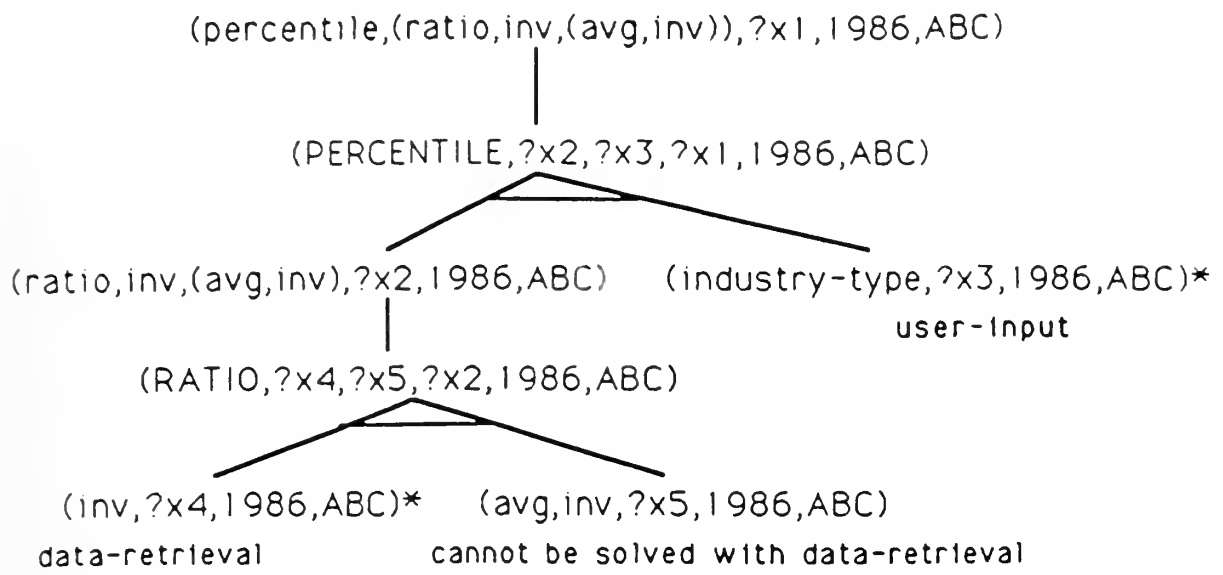


- Since all terminal nodes are unsolvable, this example is treated as a negative example. It modifies the condition of the schema as follows:

G: (percentile,(ratio,B/S-var,var2),yr,fn), or
(percentile,(ratio,var1,1/S-var),yr,fn), or
(percentile,(ratio,var1,var2),yr,fn)^(yr<1988)
S (percentile,(ratio,assets,B/S-var),1986,ABC)

Figure A-4. Applying the Schema to a Negative Example

● An example ,(percentile,(ratio,inv,(avg,inv)), ?x1,1986,ABC), with the partial pattern-matching, (inv/var1, 1986/yr,ABC/fn), has the following instantiated solution



● It then modifies the schema as follows.

condition:

G (percentile,(ratio,B/S-var,var2),yr,fn), or
(percentile,(ratio,var1,I/S-var),yr,fn),or
(percentile,(ratio,var1,var2),yr,fn)^(yr<1988),or
(percenitle,(ratio,var1,(avg,var2)),yr,fn).

S: (percentile,(ratio,assets,B/S-var),1986,ABC), or
(percentile,(ratio,inv,(avg,inv)),1986,ABC).

solution:

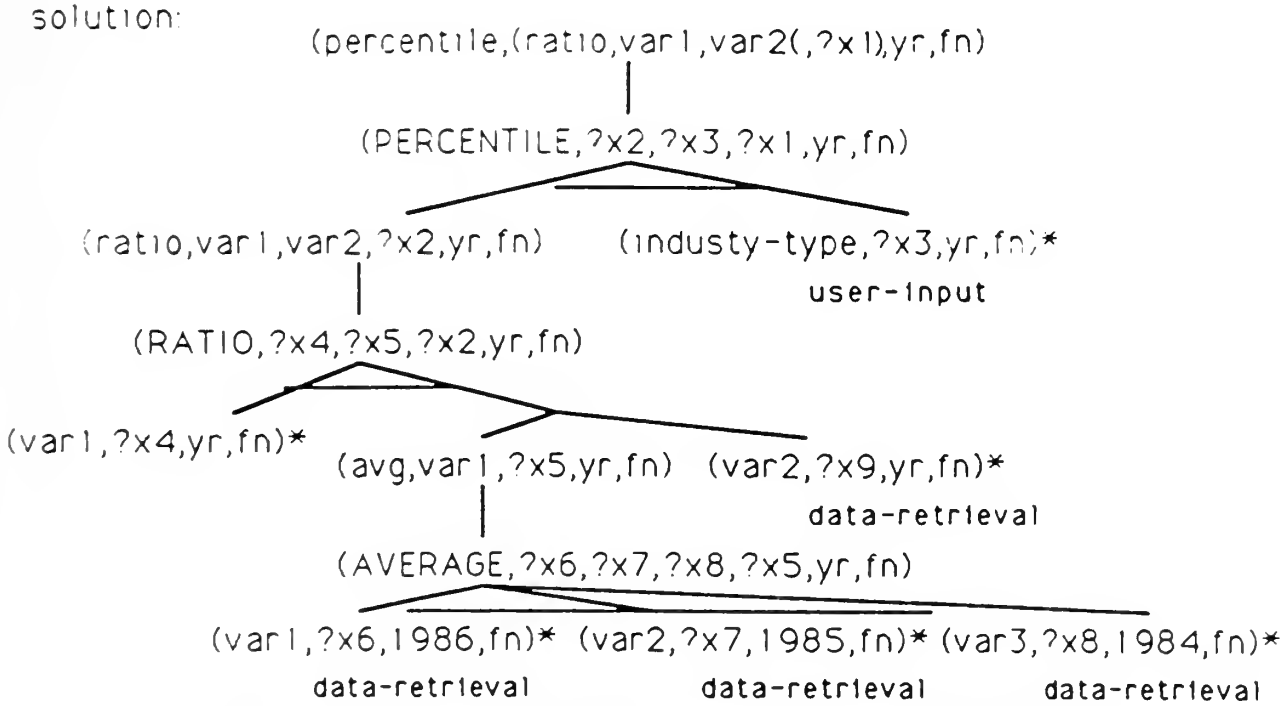


Figure A-5. Applying the Schema to a Near-miss Example

```
Explanation Structure:

grant-loan (ABC)
      | U1
      |
grant-loan (?a1) :rule1
  <--
reasonable-share-of-risk (?a1) &
|| enough-funds (?a1) &
||    sufficient-liquid-assets (?a1) &
||    ||
||    ||== financially-profitable (?a1) &
||    ||    financial-credit-rating (High, ?a1)
||    ||         || U2
||    ||         ||
||    ||    financial-credit-rating (High ?a8) :rule5
||    ||      <--
||    ||=== profitability-rating (High ?a8) &
||    ||      solvency-rating (High ?a8)
||    ||           || U3
||    || U3        ||
||    ||      solvency-rating (High ?a10) &
||    || sufficient-liquid-assets (?a10) :rule6
||    ||   <--
||    || cash-to-curr-liabilities (?x10 ?a10) &
||    ||   | industry-cash-to-curr-liabilities (?x11) &
||    ||   | greater-than (?x10 ?x11) &
||    ||   |   industry-cash+receivables/curr-liabilities (?x13) &
||    ||   |   | cash+receivables/curr-liabilities (?x12 ?a10) &
||    ||   |   | greater-than (?x12 ?x13) &
||    ||   |   |   inventories/curr-assets (?x14 ?a10) &
||    ||   |   |   | industry-inventories/curr-assets (?x15) &
||    ||   |   |   | | greater-than (?x14 ?x15)
||    ||   |   |   | |
||    ||   |   |   | | U4
||    ||   |   |   | industry-inventories/curr-assets (13%)
||    ||   |   |   |
||    ||   |   |   | U5
||    ||   |   |   inventories/curr-assets (40% ABC)
||    ||   |   |
||    ||   |   | U6
||    ||   |   cash+receivables/curr-liabilities (55% ABC)
||    ||   |   |
||    ||   |   | U7
||    ||   |   industry-cash+receivables/curr-liabilities (30%)
||    ||   |
||    ||   | U8
||    ||   industry-cash-to-curr-liabilities (50%)
||    ||   |
||    ||   | U9
||    ||   cash-to-curr-liabilities (77% ABC)
||    ||
||    || U10
||    profitability-rating (High ?a16) &
```

Figure 5.1 The Explanation Structure

```
financially-profitable (?a16) :rule7
  <--
pro-forma-pretax-profits/tangible-assets (?x18 ?a16) &
| industry-pro-forma-pretax-profits/tangible-assets (?x19) &
| | greater-than (?x18 ?x19) &
| | industry-pro-forma-net-profits/tangible-assets (?x17) &
| | | pro-forma-net-profits/tangible-assets (?x16 ?a16) &
| | | | greater-than (?x16 ?x17)
| | | |
| | | | U11
| | | pro-forma-net-profits/tangible-assets (65% ABC)
| | | |
| | | | U12
| | | industry-pro-forma-net-profits/tangible-assets (60%)
| | | |
| | | | U13
| | | industry-pro-forma-pretax-profits/tangible-assets (65%)
| | | |
| | | | U14
| | pro-forma-pretax-profits/tangible-assets (69% ABC)
| |
| | U15
| enough-funds (?a5) :rule3
|   <--
| funds-from-operations (?x5 ?a5) &
| | total-funds-for-debts (?x6 ?a5) &
| | | greater-than (?x5 ?x6)
| | |
| | | U16
| | total-funds-for-debts (?x9 ?a7) :rule4
| |   <--
| | applied-loan-amount (?L7 ?a7) &
| | | total-debt (?x7 ?a7) &
| | | | current-prime-rate (?r7) &
| | | | | plus (?x8 ?L7 ?x7) &
| | | | | product (?x9 ?x8 ?r7)
| | | | |
| | | | | U17
| | | | current-prime-rate (7.5%)
| | | | |
| | | | | U18
| | | total-debt ($10,000,000 ABC)
| | | |
| | | | U19
| | applied-loan-amount ($1,000,000 ABC)
| |
| | U20
| funds-from-operations ($50,000,000 ABC)
|
| U21
reasonable-share-of-risk (?a3) :rule2
  <--
tangible-net-worth/total-debt (?x3 ?a3) &
| industry-tangible-net-worth/total-debt (?x4) &
```

Figure 5.1 The Explanation Structure

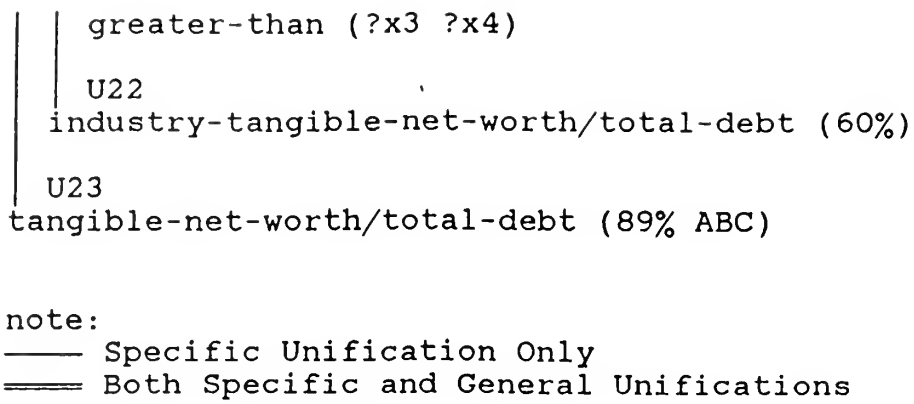


Figure 5.1 The Explanation Structure